ARCHITECTURAL PATTERN FOR SCHEDULING MULTI-COMPONENT APPLICATIONS IN DISTRIBUTED SYSTEMS

By

ABSALOM EL-SHAMIR EZUGWU

DEPARTMENT OF MATHEMATICS
AHMADU BELLO UNIVERSITY, ZARIA
NIGERIA

JULY, 2015

ARCHITECTURAL PATTERN FOR SCHEDULING MULTI-COMPONENT APPLICATIONS IN DISTRIBUTED SYSTEMS

By

Absalom El-Shamir EZUGWU
B.Sc. (ABU, 2007), M.Sc (ABU, 2011)
Ph.D./SCI/10219/2011-2012

A DISSERTATION SUBMITTED TO THE SCHOOL OF POSTGRADUATE STUDIES, AHMADU BELLO UNIVERSITY, ZARIA

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

DEPARTMENT OF MATHEMATICS
AHMADU BELLO UNIVERSITY, ZARIA
NIGERIA.

July, 2015

DECLARATION

I declare that the work in the Dissertation entitled "*Architectural Pattern for Scheduling Multi-Component Applications in Distributed Systems"* has been carried out by me in the Department of Mathematics under the supervision of Professor S. B. Junaidu, Dr. A. A. Obiniyi, and Dr. S. E. Abdullahi.

The information derived from the literature has been duly acknowledged in the text and a list of references provided. No part of this dissertation was previously presented for another degree or diploma at this or any other Institution.

Absalom El-Shamir EZUGWU	_____	_____
Name of student	Signature	Date

CERTIFICATION

This dissertation entitled "ARCHITECTURAL PATTERN FOR SCHEDULING MULTI-COMPONENT APPLICATIONS IN DISTRIBUTED SYSTEMS" by Absalom El-Shamir EZUGWU meets the regulations governing the award of the degree of Doctor of Philosophy of Ahmadu Bello University, Zaria, and is approved for the contribution to knowledge and literary presentation.

_____	_____	_____
Prof. S. B Junaidu Chairman, Supervisory Committee	(Signature)	Date
_____	_____	_____
Dr. A. A. Obiniyi Member, Supervisory Committee	(Signature)	Date
_____	_____	_____
Dr. S. E. Abdullahi Member, Supervisory Committee	(Signature)	Date
_____	_____	_____
External Examiner	(Signature)	Date
_____	_____	_____
Dr. B. Sani Head of Department	(Signature)	Date
_____	_____	_____
Prof. A. H. Zoaka Dean of Postgraduate School	(Signature)	Date

DEDICATION

This dissertation is dedicated to my lovely wife

ACKNOWLEDGEMENTS

First of all I would like to thank **God Almighty** for the grace and strength provided to me for completing this course.

I am deeply indebted to my guide, Prof. Sahalu Junaidu, for molding a Parallel and Distributed Systems researcher out of clay. The intellectual excitement during my collaboration with him is a 'high' that will last me a life time. I use the word "guide" in its truest sense when I address Prof. Sahalu. He was there, like a sign post in a desert, when I was lost exploring new frontiers. He has an uncanny sense of the right direction and a never say die attitude when faced with hard problems. He was there, like a friend, when I had personal problems. I consider myself to be truly fortunate to be a member of the "Sahalu's academic family"

Many thanks go to the following: Dr. Afolayan Ayodele Obiniyi, Dr. Marc Eduard Frincu, and Dr. Seyed Mohammed Buhari. I wish to express my deepest gratitude to Dr. Afolayan Obiniyi, for having advised me brilliantly and tirelessly during the whole process. Dr. Obiniyis' intellectual depth, originality and his "sixth-sense" for innovation have guided me and contributed enormously to my results. Thank you for being extremely supportive and providing insights into the relevance of my dissertation along facets that I never thought of. He nudged me to make my ideas more crisp and avoid ad-hocness in my writing style. Dr. Marc Frincu guided me in my research, and served as a role model and a friend. He provided deep and insightful comments at various stages of this research. Our consistent chat and discussions, despite barriers in distance, on the state of the field and directions, served to spur me on in many interesting directions. I would like to express my deep sense of gratitude to him for all this and more. Also, thanks go to him for putting learning in multi-agent systems on the research map of Multi-agent Systems and Distributed System Scheduling communities. Dr. Seyed Buhari asked me the right questions and made sure that I noticed the appropriate connections to the rest of the research community in the area of learning. He read my documents carefully and provided keen feedback. He nailed home the point that I should write to make others understand rather than to dazzle them with an array of Distributed Scheduling jargon. Thanks Dr. Buhari, for being such an influence on how I present my ideas. I would also like to acknowledge the contribution of Dr. Saleh

Abdullahi, who precious advice and strong support has provided fruitful leads in directions which I would have otherwise overlooked.

On a personal level, I would like to thank the following scholars Dr. B. Sani, Dr. A. M. Ibrahim, Dr. A. O. Ajibade, Dr. A. Yahaya, Dr. A. Mohammed, Prof. A. Alkali, Prof. D. Choji, Prof. M. Ogbe, Prof. H. E. Kwon-ndung, and Prof. E. E. Udensi, for all the support they provided. None of this would be possible without you.

Many colleagues chipped in with words of wisdom at different stages of my studies and it was a great time with them around: special mention goes to Dr. Donfack Kana, Mr. Ngene C. Nnamani, Dr. Peter Yusuf, Ofemile, Mr. Chinedu Peter, Mr. Francois Siewe, Dr. Chikwu Alerochi, Mr. Yakmut Daniel, Mr. Agushaka Jeffery, Mr. Ofem Paulinus, Mr. Ijakoli Alphonsus, Mr. Habu Peter, Mr. Eke Christopher, Mr. Adetula Emmanuel, Mr. Alaku Habila, Mrs. Dauda Priscilla, Mrs Daya Patience and Mr. Mohammed Ibrahim.

My special thanks go to Bridget Pwajok, Nneoma Okoroafor, and Aina Sikirat for having read and commented carefully on grammar in this dissertation.

Big thanks to my parents for believing in me and providing the much needed emotional support, especially when I “gave up”. I owe it all to them. Last but in no way the least, my wife Joy provided invaluable support for me in this process. She prepared me for my talks, provided comments on the draft versions of my papers, and nudged me along when I slacked off, but importantly helped build many shared tender moments that sustain one through the tougher times. To her, I am extremely grateful.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ABFF	-	Agent Based First Fit
AggressiveBF	-	Aggressive Backfilling
AI	-	Artificial Intelligence
AMQP	-	Advanced Message Queuing Protocol
ANN	-	Artificial Neural Network
ARS	-	Adaptive Random Search
AUML	-	Agent Unified Modelling Language
BJF	-	Biggest Job First
BS	-	Binary Search
BSBS	-	Binary Search Bubble Sort
CF	-	Close to File
CIS	-	Centralised Index Search
CONS	-	Conservative Backfilling
DAG	-	Directed Acyclic Graph
DAI	-	Distributed Artificial Intelligence
DIS	-	Distributed Index Search
DS	-	Distributed Scheduler
EDF	-	Earlier Deadline First
ESG-LS	-	Earlier Suitable Gap-Local Search
FCFS	-	First Come First Served
FIFO	-	First in First Out
FS	-	Flooding Search
FTT	-	File Transfer Time

GIIS	-	Grid Index Information Service
GRIS	-	Grid Resource Information Service
GS	-	Gap Search
HPC	-	High Performance Computing
HRI	-	Hop-count Routing Index
ICF	-	Improved Close to File
JSNITF	-	Job with the Smallest Number of Incomplete Task First
JSON	-	Java Script Object Notation
LFJF	-	Least Flexible Job First
LSTF	-	Limited Shortest Task First
MAS	-	Multi-agent System
MA-SP	-	Multi-agent Scheduling Policy
MDS	-	Meta Directory Service
MIMD	-	Multiple Instruction Multiple Data
MISD	-	Multiple Instruction Single Data
MPGP	-	Modified Partial Global Planning
MPI	-	Message Passing Interface
MSE	-	Mean Square Error
NN	-	Neural network
NWS	-	Network Weather Service
OOD	-	Object-Oriented Design
OODBMS	-	Object-Oriented Database Management System
ORLIS	-	Opportunistic Resource List Indexer Search
PE	-	Processing Element

PGP	-	Partial Global Planning
RI	-	Routing Indices
RMS	-	Resource Management System
SaaS	-	Software as a Service
SIMD	-	Single Instruction Multiple Data
SISD	-	Single Instruction Single Data
SNLDD	-	Sorted Node in Leveled DAG Division
STF	-	Shortest Task First
UML	-	Unified Modelling Language
UUID	-	Universal Unique Identifier

LIST OF SYMBOLS

M	-	Set of Machines
J	-	Set of Jobs
O	-	Set of operations
ps	-	Processor speed
CS	-	CPU Speed
ms	-	Memory size
np	-	Number of processors
bw	-	Associated I/O bandwidth
$\mathbb{C}_1$	-	Type of machines
$\mathbb{C}_2$	-	Number of machines
C	-	Denotes number of clusters
p_{ij}	-	Processing time
R	-	Set of resources
Q	-	Denotes a Queue
A	-	Set of agents collaborators
$\beta_1, \beta_2, \beta_3, \beta_4$	-	Assigned Weights to resource parameters
CL	-	Cluster
pf	-	Performance function
pr	-	Performance ration
H	-	Resources capacity

ABSTRACT

Scheduling in distributed systems generally aims to leverage the power of diverse heterogeneous, geographically distributed, multiple-domain-spanning computational resource to provide optimal system performance, high throughput computing and maximum resource utilization. To achieve this goal, an efficient and effective scheduling system is fundamentally important. However, it is very difficult to do this using the traditional off-the-shelf scheduling software. Many issues have to be treated, such as, poor system design, limitation in heuristic-based implementation, system adaptability and scalability. *This leads to the necessity of an additional software infrastructure to provide solutions to the aforementioned problems.* This research focuses on the design of a general-purpose distributed scheduling framework, based on the concepts of object-oriented and multi-agent design patterns. The dissertation describes how object-oriented and multi-agent design patterns are used to model an adequate scheduling system. Specifically, a multi-component scheduling framework, capable of scaling the scheduling of user applications across different distributed system environments is proposed. The proposed solution suggests using two levels of scheduling: global and local. The global scheduling policies assign the response time requirements for scheduling component service invocations. The local scheduling policies are responsible for performing request scheduling, in order to meet these requirements. The proposed scheduling approach does not require a central point of control, its platform independent, and essentially, it provides quality of service to both user applications and resource owners, by reaching a compromise between their necessities. The experiments conducted using simulation, were used to study the effectiveness and the feasibility of the proposed scheduling schemes in respect to various deployment requirements. The validity of the simulation was confirmed by comparing its results with the results obtained in experiments with other heuristic-based scheduling algorithms. The proposed approach was shown to work well for different classes of applications flow, including, compute-intensive and data-intensive applications, under different scheduling policies.

CHAPTER ONE

GENERAL INTRODUCTION

1.1 Background of the Study

Research in distributed systems, specifically, grids and clouds scheduling has grown tremendously over the last four decades. Scheduling problems are complex, and as such, several individuals and research groups from different fields have studied them for decades. Different scheduling techniques have emerged as a result, and some have been applied to tackle many different scheduling problems. Some of these scheduling techniques include the critical path methods, dynamic programming, branch and bound, and priority rules. Unfortunately, most of the scheduling problems have no known polynomial time algorithms, and are therefore classified as NP-hard in complexity theory. Based on this reason, many researchers have used different scheduling methodologies such as the artificial intelligence, expert systems, operations research, multi-agent, object-oriented and sometimes, a combination of two or more of these methods to develop scheduling infrastructures.

Generally speaking, a large number of the scheduling systems developed and in use today are basically application specific type of systems, designed specifically to solve specific problems, while only a handful are for general-purpose. Often, the general-purpose systems lack adaptability in terms of platform shift, and are usually not cost effective to maintain due to initial poor design conception and implementation. On the other hand, application specific scheduling systems usually fare somewhat better than their generic counterpart, for the simple fact that they tend to be more precise in handling problems. However, they are sometimes more difficult to manage and cannot adapt to dynamic environments. Considering these facts, it then justifies the fact that

there should not be any trade-off whatsoever between adaptability and performance with regards to adequate and quality design, implementation and adaptability.

In this dissertation, the design of an inter-provider distributed scheduling platform that addresses the scheduling of multi-component applications within the context of distributed computing systems environment is considered. The projected problem domain to be addressed is targeted at allocation and coordination of computational tasks scheduling problems in heterogeneous computing platforms such as Cloud computing, Grid computing, Utility computing, Client-server computing, Cluster computing and Peer-to-peer computing.

A carefully designed scheduling pattern can be used to solve some of the challenging problems identified in Section 1.3. However, the dissertation main focus is to addressing three main issues, which include, ***system design paradigm*** (a major reason for the limited success of the current scheduling systems, designed based on the concepts of classical design techniques which are either operation or data oriented)**,** ***heuristic-based implementation methods*** (current scheduling systems or scheduling algorithms use heuristics or approximation methods to find solutions to most scheduling problems) ***and system adaptability*** (current scheduling systems are mainly used for meeting specific needs and objectives, and for solving specific domain related problems)**.** Generally, a complete scheduling system has to provide capabilities for **scaling** of individual applications across variety of distributed system environments, **extensibility** (perform similar scheduling task in a different distributed system platform), **robustness** (fault-tolerance in a dynamic heterogeneous and unpredictable distributed computing environment), **adaptability** with respect to inter-providers' tasks coordination and good performance given limited but available scarce resources.

The remainder of this chapter present some fundamental concepts and discusses in detail the research motivation, proposed research objectives and concepts, ways of achieving the set objectives and the contribution of this dissertation to the areas of distributed tasks scheduling in the domain of distributed computing systems.

1.1.1 Fundamental Concepts

The scheduling problem in distributed systems environment has been an active research topic, and therefore, many terminologies have been suggested. Unfortunately, some of the terms are neither clearly stated nor consistently used by different researchers, which frequently confuse readers. For clarity, in this dissertation, some key terminologies are re-defined (adopted from (Malathi and Sarumathi, 2010; Zhang *et al.*, 2009; Zhu and Ni, 2013)).

Distributed system: A distributed system is a networked framework that enables the sharing, selection, and aggregation of geographically distributed autonomous and heterogeneous resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements. Examples of these include grids, clouds, client-servers, peer-to-peer, clusters and workstations.

Resource: A resource is an infrastructure that is required to execute a task, for example, a processor used for data processing, data storage device, or a network link for data transporting.

Scheduling: Scheduling is a technique with which scheduler gathers both the resource and application state information, selects appropriate resources, generates schedules, predicts the potential performance for each candidate schedule, and determines the best schedule for the applications to be executed on a distributed system, subject to some performance goals.

Scheduling goals: The main goals of scheduling are to optimize system and user oriented objectives such as maximizing throughput, maximizing resource utilization, minimizing execution time, and fulfilling economic constraints.

Scheduler: A scheduler is the mediate resource controller that interfaces between the users' applications and the underlying resources. The scheduler is responsible for discovering available resources for an application, selecting the appropriate system(s), and submitting the application.

Application: An application is a collection of jobs coordinated to solve a certain problem. In other words, an application may consist of a number of jobs, either dependent or independent, which together fulfil the whole task.

Multi-component applications: may be defined as large, related collection of interacting computational tasks, schedulable across multiple execution sites and designed to achieve a specific overall result.

Job: A job is considered as a single unit of work within an application. It is typically allocated to execute on one single resource. It has input and output data, and execution requirements in order to complete its task.

Task: A task is an atomic unit of a job to be scheduled by the scheduler and assigned to a resource.

Site: Site is considered as a computational resource that can interact with schedulers directly, and accept workloads from the schedulers. A site may be a simple personal machine, a workstation, a supercomputer, or a cluster of workstations. From a scheduler's point of view, a site is an atomic resource unit that can be allocated to application jobs (Fig. 1.1).

Processing node: A processing node (also referred to as computing node or computational node), means different things under different environments. Under the context of Grid environments, a processing node is a site. However, in the context of Cluster environments, a processing node means a stand-alone computer.

Multi-component computing: Multi-component computing is the seamless application of geographically distributed non-dedicated meta-computing systems to user applications. A meta-computing platform generally consists of a collection of heterogeneous, non-dedicated computing resources, which may include multiprocessor platforms, computer Grids, cloud computing infrastructures, among others.

Hardware architecture: Computer architecture can be classified into four categories in terms of their control flows and data flows. *SISD:* an acronym for single instruction single data, a single instruction flow operates on a single data flow. This describes the sequential processing techniques, *SIMD:* stands for single instruction multiple data, a single instruction flow on multiple data flow simultaneously, *MISD:* multiple instruction single data, which implies multiple instruction flows operate on single data flow simultaneously, it seldom appears in real architecture and *MIMD:* multiple instruction flows operate on multiple data flows simultaneously. The two most commonly used architectures are the *SIMD* and *MIMD*.

Heterogeneous Computing Environments: are categories of distributed computing resources. They comprise of several Processing Elements (PEs), each with different processing capabilities or speeds (in-terms of how fast they can execute tasks), communication latencies between PEs, and the amount of memory associated with each PE. The main focus of this dissertation, however, is on heterogeneous computing environments and is explained in detail, in the next chapter.

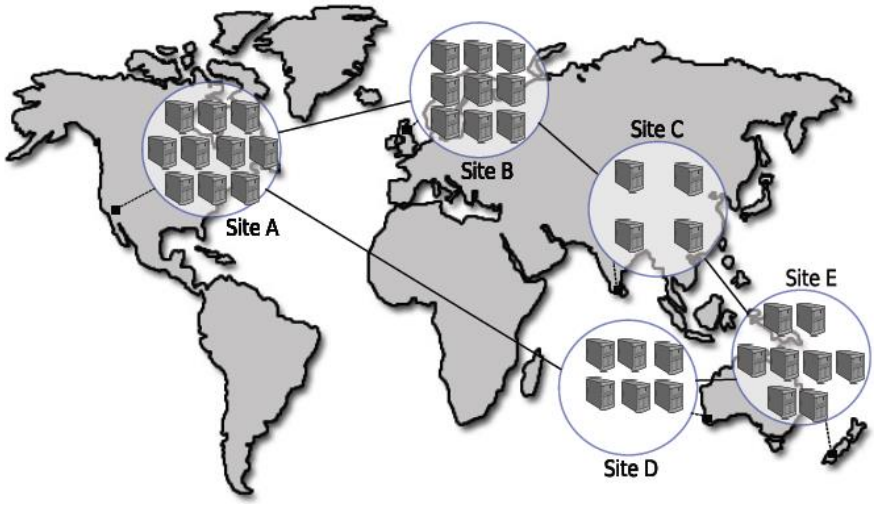


Fig. 1.1: Computing Environment where Clusters of PEs from Five Sites are Connected by High-Latency Networks (Janjic, 2012)

1.2 Research Motivation

The need for distributed computing platforms is on the rise, especially with the recent advent of cloud computing and the Big Data paradigm. For example, in the work of Konwinski (2012), it was stated that the clusters of commodity servers have been widely adopted as primary computing platform for large Internet services, data-intensive scientific applications and enterprise analytics. These distributed computing environments are somewhat defined and characterised by heterogeneity and dynamism. According to Frincu (2011), the behaviour of most scheduling algorithms designed specifically for this environment is drastically and continuously being affected by the system's configuration. Subsequently, these scheduling algorithms become susceptible to other challenging issues such as fault-tolerance, scalability, computational and communication heterogeneity and resource sharing. These issues will be discussed in detail in chapter two. Again, sometimes the negative performances of these algorithms have resulted in poor execution of user submitted tasks and sometimes non execution of such tasks might be recorded. One noticeable consequence of these effects is that, often

times, wrong estimate of the scheduling algorithm is taken and it negatively influences the cost function (such as, execution cost and processing time).

The problem of scheduling distributed heterogeneous resources and computational tasks which are somewhat compute intensive is known to be extremely challenging or is said to be NP-complete and has been studied for decades (Stavrinides and Karatza, 2012; Wu *et al.,* 2011). NP-complete problems are in NP, the set of all decision problems whose solutions can be verified in polynomial time; NP may be equivalently defined as the set of decision problems that can be solved in polynomial time on a non-deterministic Turing machine. The complexity of this problem increases especially when the scheduling process becomes dependent upon the activities of some other scheduler that lacks knowledge of the applications to be scheduled. This type of scheduling process is common within distributed scheduling environments, where for example, scheduling agents have to cooperate, coordinate and negotiate among themselves to achieve particular scheduling goals, such as, maximizing resource utilization and determining an allocation which satisfies certain restrictions.

In line with the identified scheduling challenge, the problem can be addressed, by drawing upon the potentials of object-oriented software design, and multi-agent technology to deal with such complex situations. There are many advantages in following an object-oriented design approach in the development of a scheduling system (Pinedo and Yen, 1997), also, it seems appropriate to consider multi-agent scheduling approach as good candidate for solving dynamic scheduling problems, especially in a typical distributed heterogeneous computing system environment. One good reason for this suggestion is that, agents based on some special characteristics are capable of handling complex problems in distributed systems (Jennings, 2001; Lucena and Nunes, 2012).

Pinedo and Yen (1997), are of the opinion that, in choosing an object-oriented design approach in the development of distributed scheduling system, the method offers some specific and unique benefits. First, the design should be modular, which makes maintenance and modification of the system relatively easy. Second, large segments of the code should be reusable. This implies that two scheduling systems that are substantially different may still share a significant amount of code. Third, the designer should think in terms of the behaviour of objects, not in low-level detail. In other words, the object-oriented design approach can speed up the design process and separate the design process from the implementation process. It is equally important to note that, by following this method, the proposed scheduling solution would have automatically addressed the major point raised in the dissertation' problem statement regarding design and partly implementation issues mentioned in section 1.3.

The dissertation' major goal is *to develop an architectural design pattern for a broad-spectrum scheduling system that addresses some of the key scheduling problems in a typical distributed computing environment.* Broad-spectrum in this case, implies a scheduling system that is adaptive and somewhat generic solution approach to problem solving and whose tasks allocation and coordination capability is able to scale beyond certain inter-provider environment. Multi-Agent Systems offer a natural extension to distributed tasks scheduling as they allow resources from multi-providers to inter-operate autonomously through negotiation (Ouelhadj *et al.*, 2005).

In a typical distributed problem solving environment scenario, intelligent agents often tend to pursue their own individual goals when solving a problem. However, most multi-agent schedulers rely on specialized agent hierarchies (Cao *et al.*, 2005), that could be likened to the conventional centralized schedulers which are disadvantaged by the single point of failure orientation that often times leads to failure in the overall

functioning of the scheduler. Besides, they require for each virtual organisation to use the existing agents and thus to redefine their scheduling policies. In contrast, a completely decentralized approach with no strict adherence to any form of hierarchies and in which agents are autonomous in their dealings, offers a more fault-tolerant scheduling environment. Likewise, in order to allow virtual organisations to maintain their autonomy, only the communication protocol should be standardized. In this way, any member of the distributed system could expose its own custom built agent, in terms of scheduling policies, to deal with task coordination and negotiations issues.

Based on the aforementioned facts regarding the different scheduling system implementation, the dissertation also recognizes several approaches that have been proposed towards creating either adaptive scheduling strategies or multi-agent system for computational tasks scheduling. However, most of these approaches are problem-defined and often focus on individual scheduling knowledge orientation such as coordination, negotiation, service-level agreement, adaptive scheduling, interactive scheduling, and cooperative scheduling. In this sense, building a functional Scheduling Platform which is fully distributed in terms of communication, storage, scheduling and negotiation solutions and also, adaptive by nature is obviously still an open research challenge. To address this challenge, a proposal is made in this dissertation, *to design an adaptive inter-provider scheduling framework based on the principle of object-oriented and autonomous multi-agent software design patterns. The proposed framework environment is modelled upon a decentralized peer-to-peer network overlay*. This choice is simply based on the fact that, this network model often produces efficient systems interaction and coordination (Crespo and Garcia-Molina, 2005; Solar *et al.,* 2012).

1.3 Research Problem Description

The efficient scheduling of available computing resources can greatly increase the amount of resources available to users of data-intensive applications, where the time spent on file transfer dominate the computing times. In the case of compute-intensive applications, the situation is the opposite. However, it is very difficult to achieve this by using the traditional off-the-shelf scheduling software and resource provisioning systems. Many issues have to be addressed first, such as *system design paradigm, heuristic-based implementation methods, scalability, application deployment, distributed scheduling, adaptability, heterogeneity, unpredictability of the resource environment, fault-tolerance, and system flexibility*. This leads to the necessity of an additional software infrastructure to provide solutions to the earlier mentioned challenges in this section. The following properties must be considered when developing such an infrastructure.

a. It must allow the provisioning of heterogeneous resources efficiently, such as, processors, memory, storage, and other kinds of hardware resources, which would otherwise remain idle or underutilized.
b. It needs to allow for decentralized scheduling and execution of users' jobs on various heterogeneous resources running on different distributed computing environments.
c. It needs to hide low level scheduling details from the user. The user does not need to know where submitted applications will be executed. Object-oriented concept is well suited for this approach.
d. It needs to provide quality of service to both users' applications and resource providers, by reaching a compromise between their requirements.
e. It needs to be adaptable, by allowing existing applications to easily adapt to the new context.

f. It needs to have a low deployment cost, not requiring extensive reconfigurations on existing systems.

g. It must not rely only on heuristic methods alone for solution to problems. Autonomous agents can also be trained using non-linear techniques to perform different scheduling functions with greater precision. Even though heuristic approaches can be fast and efficient, agents are more likely to be adaptive, scalable and fault-tolerant than most approximation algorithms.

Generally, an adequate general-purpose distributed scheduling system should overcome these needs or challenges to leverage the promising potential of any of the distributed systems, providing high-performance services.

There are instances of many projects that claim to have addressed the resource or data-intensive scientific applications scheduling problem. Systems such as Globus (Foster, 2006), Netsolve, (Casanova and Dongarra, 1997), Legion (Chapin *et al.*, 1999), the emergent cloud computing Schedulers (Nurmi *et al.*, 2009; Buyya *et al.*, 2009), and OpenNebula (Sotomayor *et al.*, 2009), present resource management architectures that support resource discovery, dynamic resource status monitor, resource allocation, and job control. These architectures make it easy to create a high-level scheduler. Legion also provides a simple, default scheduler. However, they usually rely on knowing the state of every computational resource, thereby paying much attention to low-level scheduling information and neglecting individual application requirements. But Dail *et al.* (2000), show that schedulers with special knowledge of the individual application can easily outperform a default scheduler that lacks this knowledge.

Equipping schedulers with too much static information limits their scalability, as in the case of Condor (Tannenbaum *et al.*, 2002), and supplying insufficient information limits

their scheduling capabilities, as in the case of BOINC (Anderson, 2004). These problems can be overcome with a decentralized solution that relies only on the dynamic resource information coming from the resource provider end (example: available number of processor per node, processor speed, memory, and associated bandwidth) and the individual application resource requirement (example: total memory and minimum bandwidth). However, under dynamic scenario, supplying schedulers with fine-grained application and resource description, guide the scheduler to obtain much better matches with the underlying resources, and hence more accurate performance predictions.

Other additional works that have addressed the aforementioned issues to some certain degree include (Kim *et al.*, 2008; Basu *et al.,* 2009a; Rahman *et al*., 2010, and Anderson, 2004). However, none of these projects addressed the scheduling problem from the perspective of both application and resource description granularity simultaneously.

1.4 Research Aim and Objectives

The main **aim** of this research is to develop a broad-spectrum scheduling system architectural design pattern, that supports the scheduling of diverse multi-component applications in a distributed computing system environments. Similarly, the research objectives are to:

i. Propose a general-purpose scheduling framework which consists of a resource model, an application model, a performance model and a scheduling policy based on the concept of object-oriented and multi-agent design patterns.
ii. Develop a Neural Network based Multi-agents intelligent resource selection framework that provides a common resource selection service for different kinds of application.

The design goal in this case is to provide quality of service (QoS) to both users' applications and resource owners, by reaching a compromise between their necessities.

iii. Propose novel job forwarding and resource discovery methods, based on the concept of peer-to-peer computing paradigm. Here, the concept of routing indices document discovery technique is introduced to address the job query forwarding and resource discovery problems, in a typical multi-agent distributed scheduling system environment.

iv. Implement and evaluate the developed architectural framework using realistic application scenarios to support the applicability of the proposed scheduling structure in real-world scheduling problems.

A general-purpose scheduling framework suitable for all kinds of distributed systems environment comprises four models: application model, resource model, performance model, and scheduling policy. An application model extracts the characteristics of applications to be scheduled from the job description file. A resource model describes the characteristics of the underlying resources inside the distributed systems. A performance model is responsible for predicting the performance potential of a schedule. The prediction is usually based on the prediction of the behaviour of a specific job on a specific computational resource. Scheduling policy is responsible for deciding how applications should be executed and how resources should be utilized.

1.5 Methodology

In this dissertation, the focus is to understand the existing state-of-the-art scheduling systems in distributed computing system environments, and then, apply that understanding to design and simulate a better general-purpose scheduling system. However, throughout the course of this study, the following research processes were applied to achieve the dissertation research objectives:

a. Review related literature and state-of-the-art in distributed scheduling systems.
b. Study the concept of design patterns in object-oriented software development and multi-agent technology.
c. Use the insight gained from (a) and (b) to develop a general-purpose scheduling framework.
d. Study some mathematical concepts within the scope of basic set theory and apply this knowledge to model resource selection, configuration, and mapping.
e. Study communication patterns in peer-to-peer system and apply the same concept in the agent interaction for the proposed scheduling system.
f. Implement the general-purpose scheduling framework developed
g. Use a standard benchmark workload to simulate and evaluate the proposed scheduling models on a targeted distributed system framework.

1.6 Research Contribution to Knowledge

The main contribution of this research is the *conceptualization and application of object-oriented design patterns and multi-agent technology to the design of a general-purpose scheduling model*. Then use the general model to propose a novel, more decentralized, distributed scheduling architecture, which addresses the limitations of the existing technologies. An intelligent resource selection framework that provides a common resource selection service for different kinds of applications is also presented. In general, the contributions are summarised as follows:

a. Presentation of a framework model for a decentralized general-purpose scheduling architecture, which addresses the issues of poor system design, heuristic-based implementation and limited system scalability.

b. Presentation of a scheduling design pattern architecture, that shifts scheduling logic to intelligent agents.
c. Design and simulation of an intelligent resource selection framework that maximizes resource utilization and performance of distributed systems.
d. Implementation and evaluation of the proposed algorithmic framework to compare its scalability and versatility with previous researches.

1.7 Dissertation Outline

The rest of this dissertation is organized into six chapters. In Chapter two, review on related literatures which includes background information, scheduling algorithms, resource management systems, scheduling methods, object-oriented software design patterns, object-based storage system, multi-agent technology and peer-to-peer communication models are presented.

In Chapter three, a description of the proposed scheduling model in terms of objects and methods is presented. An object-oriented design pattern for the scheduling model and the translation of the scheduling components into autonomous multi-agent architectural framework is also presented. The Chapter also covers description on the communication patterns used by scheduling agent types to convey messages across different scheduling layers using a decentralized peer-to-peer network overlay infrastructure. The last section summarizes the achievements made so far.

Chapter four describes models for expressing resource selection, configuration, and mapping using multi-agents coordination process. A training pattern for the scheduling agent types using Artificial Neural Network is discussed. Subsequently, a description of the general-purpose resource selection model, performance model and mapping strategy of the cluster application used in the dissertation case study are presented.

In Chapter five, the implementation of the general-purpose resource selection framework and the agent-based scheduling system is achieved, by simulating each stage in the scheduling process. The research findings, results and discussions are also discussed in this chapter. Finally, the work is concluded and the future work is presented in Chapter six. This last chapter is followed by the list of references cited through the dissertation and a list of author's publications.

CHAPTER TWO

LITERATURE REVIEW

2.1 Synopsis of Chapter

In this chapter, a closer look is taken at the related work done in the domains of scheduling theory, and the target environment, specifically, distributed systems. The state-of-the-art regarding scheduling and resource allocation was also investigated. Finally, the rest of this chapter is organized as follows. Section 2.2 presents the background information of related research in the areas of distributed computing systems and scheduling. In section 2.3, some studies on multi-agent systems comprising multi-agents characteristics, agents communication, design methods for multi-agent system, multi-agent based distributed scheduling architectures, multi-agent advantages and challenging issues facing the implementation of the multi-agent system are presented. In Section 2.4, the concept of object-oriented software design principles are studied. A summary of different scheduling methods and approaches are reviewed in Section 2.5. State-of-the-art scheduling systems are discussed in Section 2.6. Survey of related approaches and contributions from related dissertation are analysed in Section 2.7. Finally, Section 2.8 summarises the analysis of related work by discussing the limitations of current research and by presenting the challenges that define the research contributions, in line with the goals set in the introductory chapter.

2.1.1 Formalism Description

This section presents the mathematical models for describing scheduling problems based on the model presented in (Bruckner, 2007). Other related notations in the field of scheduling are introduced as well. The mathematical models serve as guides to the

design of the proposed scheduling systems, which involves the integration of Object-oriented and Multi-agent scheduling methods.

2.1.2 Job Characteristics

Let $M = \{M_1, M_2, \dots, M_n\}$ be the set of m machines, which have to process i set of jobs represented by $J = \{J_1, J_2, \dots, J_m\}, i = 1,2, \dots m$. A job $J_i \in J$ consists of a number of n_i operations $O_{i1}, O_{i2}, \dots, O_{i,n_i}$. Associated with operation O_{ij} is a processing requirement p_{ij}. If job J_i consists of only one operation ($n_i = 1$), then J_i is identify with O_{i1} and denote the processing requirement by p_i.

Therefore, let assume that each job $J_i \in J$ consists of n_i set of operations, which is represented as $O = \{O_{i1}, O_{i2}, \dots, O_{i,n_i}\}$. To every operation $O_{ij} \in O$, a process requirement p_{ij} is associated (Bruckner, 2007). In the particular case in which job $J_i \in J$ is made up of only $O_{i1} \in O$, J_i is identify with O_{i1}. Each operation O_{ij} is identified with a set of machine $\mu_{ij} \subset M$ that can either be dedicated or parallel (Frincu, 2011). In order to generalise the problem, let consider that an operation O_{i1}can be processed on any machine capable of solving it, in which case the machine is called a multi-purpose machine (Bruckner, 2007). It is also possible for operation O_{i1} to simultaneously use all the machines in μ_{ij} in which case we deal with multiprocessor task scheduling.

In the proposed work, a set of characteristics are added to each machine $m_i \in M$ and represented as $M_j^{'} = \{ps, ms, bw, np\}$, where ps stands for the processing speed, ms stands for the memory size, bw stand for the associated network bandwidth and np represents the number of processors attached to each machine. It is equally, important that some of these characteristics are likely to vary over time and may be required to be updated periodically.

2.1.3 Machine Environment

The machine environment is characterised by a string $\alpha = \alpha_1\alpha_2$ of two parameters, where the possible values are $\circ, P, Q, R, PMPM, QMPM, G, X, O, J, F$ for α_1 and $\circ$ or any positive integer for α_2. Where $\circ$ denotes the empty symbol (thus, $\alpha = \alpha_2$ if $\alpha_1 = 1$), then each job $j_i \in J$ consists of a single operation. Parameter α_1 points out the number of machines in the machine environment, while α_2 indicates the type of the machines as follows:

a. If $\alpha = \circ$, each job must be processed on a specified dedicated machine.

b. If $\alpha \in \{P, Q, R\}$, then this denote parallel machines, that is each job $j_i \in J$ can be processed on each of the machines $M_1, M_2, \dots, M_m$. Three possible classes of parallel machines can be taken into consideration, in this case: First, is the identical parallel machines expressed by setting $\alpha = P$. Thus, for the processing time p_{ij} of job j_i on machine m_j, $p_{ij} = p_i$ for all machines m_j. The second is the uniform parallel machines, which is expressed by setting $\alpha_2 = Q$, and the processing time is $p_{ij} = p_i/s_j$, where s_i is the speed of the machine m_j. Finally, the third class of machine is the unrelated parallel machines denoted by $\alpha_1 = R$, which processing time is expressed as $p_{ij} = p_i/s_{ij}$, where s_{ij} represents the job-dependent speed of machine m_j.

c. If $\alpha_1 = PMPM$ and $\alpha_1 = QMPM$, then this denote multi-purpose machines with identical and uniform speeds, respectively.

d. If $\alpha_1 \in \{G, X, O, J, F\}$, this represents a multi-operation model, that is associated with each job j_i. In this case, there is a set of operations $O = \{O_{i1}, O_{i2}, \dots, O_{i,ni}\}$ that is identified with a set of dedicated machines μ_{ij}. Furthermore, there are precedence

relations between arbitrary operations. This general model is called a general shop. Depending on the value of α_1, the following multi-operational model are given: general shop ($\alpha_1 = G$), Job shops ($\alpha_1 = J$), flow shops ($\alpha_1 = F$), open shops ($\alpha_1 = O$), and mixed shops ($\alpha_1 = M$). In a job shop, for instance, there is a special precedence relations of the form $O_{i1} \rightarrow O_{i2} \rightarrow \ldots \rightarrow O_{i,ni}$ for $i = 1, 2, \ldots, n$

Furthermore, it can be generally assumed that, $\mu_{i,j} \neq \mu_{i,j} + 1 \; for \; j = 1, 2, \ldots, n_i - 1$. We call a job shop in which $\mu_{i,j} = \mu_{i,j} + 1$ is a possible job shop with machine repetition. The flow shop, open shop and mixed shop are all particular cases of the job shop.

2.1.4 The Notion of Resources

In this work, a resource is introduced to express the attributes associated with each machine, a network node or a cluster. Let $R = \{R_1, R_2, \ldots, R_m\}$ be a set of resources, where in general, $R_k = \{\cup M_j | M_j \in M\} \subseteq M$. Also, let there exists a queue $Q = \{Q_1, Q_2, \ldots, Q_l\}$ with one-to-one or one-to-many mapping of sets of jobs given by $J = \{J_1, J_2, \ldots, J_i\}$ on sets of resources given by $R = \{R_1, R_2, \ldots, R_j\}$, then a schedule can be defined as follows:

Definition: A schedule for each job $J_i \in J$ can be defined as an allocation of one or more time intervals to one or more resources $R_i \in R$.

2.2 Background Information on Distributed Systems

Scheduling literature covers much ground and myriad of aspects. However, before discussing these various aspects, it would be necessary to be acquainted first with some of the background information and concepts that are intended to be used later in the

remaining part of our work. Having done so, a detailed discussion on the scope and content of the reviewed related scheduling literatures is presented.

2.2.1 Distributed systems

Networks of computers are everywhere. The Internet is one typical example, as are the many networks of which it is composed. Mobile phone networks, corporate networks, factory networks, campus networks, home networks, in-car networks, planetary networks; all of these, both separately and in combination, share the essential characteristics that make them relevant subject for study under the heading distributed systems. In Coulouris *et al.* (2012), a distributed system is defined as one in which software or hardware components located at networked computers communicate and coordinates their actions only by passing messages. In other words, a distributed system could be seen as system that consists of a collection of two or more independent computers which coordinate their processing through the exchange of synchronous or asynchronous message passing. Similarly, in Jackson *et al.* (2012) a distributed system is defined as a collection of agents in which resources, information, knowledge, capability, expertise, or authority are distributed and made accessible by means of networked connectivity. These simple definition covers the entire range of systems in which networked computers can usefully be deployed.

Networks of distributed computers can be logically or physically separated by any distance. The separation can span across countries, continents, in the same building or in the same room. The aforementioned definition of distributed systems has the following significant consequences as described in Coulouris *et al.* (2012) and (Eliassen, 2010).

a. **Concurrency**: In a network of computers, concurrent program execution is the norm. System users can do their work remotely on the computer without interference from other users concurently, while sharing resources such as web pages or files when necessary. The capacity of the system to handle shared resources can be increased by adding more resources (for example, computers) to the network. The coordination of concurrently executing programs that share resources is also an important and recurring topic (Coulouris *et al.*, 2005).

b. **Absence of global clock**: When programs need to cooperate, they coordinate their actions by exchanging messages. Close coordination often depends on a shared idea of the time at which the programs' actions occur. However, it turns out that there are limits to the accuracy with which the computers in a network can synchronize their clocks – there is no single global notion of the correct time (Iyer and Marculescu, 2002; Sinopoli *et al.*, 2003). This is a direct consequence of the fact that the only communication is by sending messages through a network.

c. **Independent failures**: All computer systems can fail, and it is the responsibility of system designers to plan for the consequences of possible failures (Schroeder and Gibson, 2007). Distributed systems can fail in new ways. Faults in the network result in the isolation of the computers that are connected to it, but that does not mean that they stop running. In fact, the programs on them may not be able to detect whether the network has failed or has become unusually slow. Similarly, the failure of a computer, or the unexpected termination of a program somewhere in the system (a crash), is not immediately made known to the other components with which it communicates. Each component of the system can fail independently, leaving the others still running (Balakrishnan *et al.*, 2003; Drost *et al.*, 2011).

The prime motivation for constructing and using distributed systems stems from a desire to increase computational power and efficiently share available resources (Attiya and Welch, 2004). The term ‘resource’ is a rather abstract one, but it best characterizes the range of things that can usefully be shared in a networked computer system. It extends from hardware components such as disks and printers to software-defined entities such as files, databases and data objects of all kinds.

Types of distributed systems

Distributed systems can be further classified into distributed computing systems, distributed information systems and distributed embedded systems. However, our focus and concentration will be on distributed computing systems. An important class of distributed systems is the one used for high-performance computing tasks, classified as high-performance or parallel computing (HPC). In the simplest sense, parallel computing or HPC is the simultaneous use of multiple compute resources to solve a computational problem (Kumar *et al.*, 1994; Darriba *et al.*, 2012; Hsu and Feng, 2005; Shan *et al*., 2008).

Roughly speaking, one can make a distinction between two subgroups (clusters and Grids). In cluster computing the underlying hardware consists of a collection of similar workstations or PCs, closely connected by means of a high-speed local-area network. In addition, each node runs the same operating system. The situation becomes quite different in the case of grid computing. This subgroup consists of distributed systems that are often constructed as a federation of heterogeneous computing systems, where each system may fall under a different administrative domain, and may be very different when it comes to hardware, software, and the deployed network technology.

Depending on its design objectives, Grid computing (Fig. 2.1) can be divided into: Computational grids, Data grids and Service grids (Frincu, 2011). Computational grids offer high aggregate computational resources for executing large number of tasks in a short amount of time (Augonnet *et al.*, 2011). They can also be divided into Distributed Supercomputing, High Throughput Systems or HPC and Cloud computing, recently clouds have been used as HPC platforms (He *et al.*, 2010; Gupta, and Milojicic, 2011). Data Grids provide an infrastructure for processing and storing large amount of data, stored in either a distributed file system or a large network. Service Grids offer access to resources provided by more than one machine. A recent trend in this direction is represented by the Software as a Service (SaaS) paradigm, which allows access to virtual resources by using services. SaaS is part of what is currently called Cloud computing today.

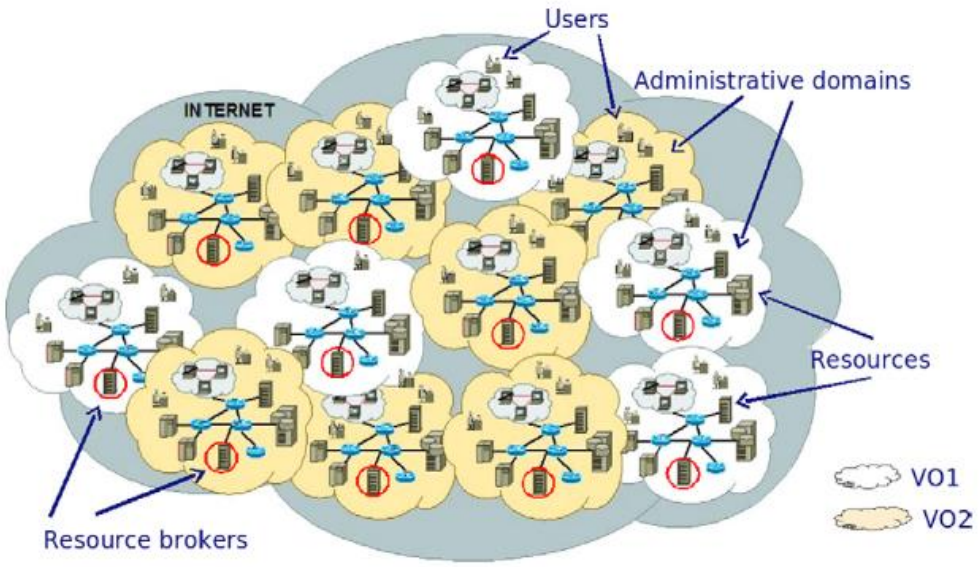


Fig. 2.1: Typical Grid Computing Setup (Caminero *et al.*, 2011)

Cloud computing presents to end user modalities to outsource on-site available services, elastic computational facilities, or data storage to an off-site, location-transparent

centralized facility or Cloud (Moschakis and Karatza, 2010). A "cloud" implies a set of machines and web services that implement cloud computing.

From the cloud computing perspective, it is assumed that, the computer can no longer be thought of in terms of the physical enclosure, that is, the server or box, which houses the processor, memory, storage and associated components that constitute the computer. Instead the "computer" in the cloud ideally comprises of a virtualized, scalable computing resources such as processors, memory, network bandwidth and storage, potentially distributed virtually across server and geographical boundaries which can be organized on demand into a dynamic logical entity. Fig. 2.2 presents an abstract architectural overview of a typical cloud computing framework.

The conceptualization of cloud resource is being empowered to elastically scale in real-time in order to assure the desired levels of latency sensitivity, performance, scalability, reliability and security to any application that runs in it (Buyyaa *et al*., 2009; Sarathy *et al*., 2012). What is truly enabling this transformation today is the virtualization technology – more specifically hardware assisted server virtualization.

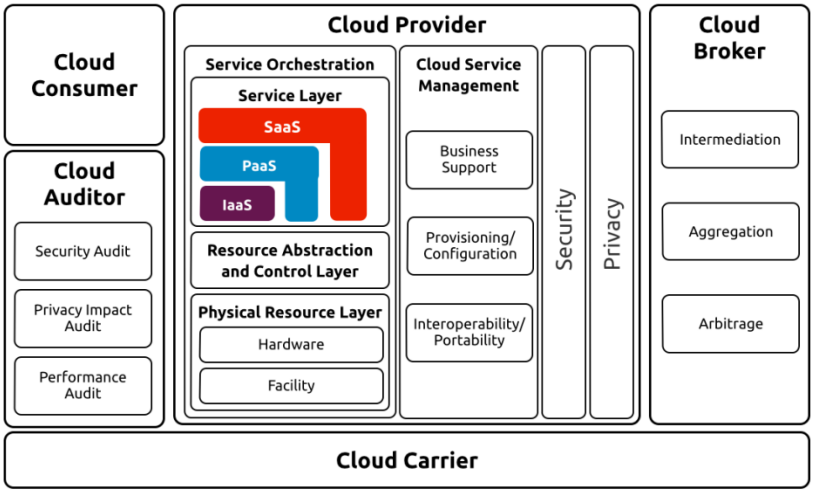


Fig. 2.2: Cloud Computing Reference Architecture (Buyya *et al*., 2009)

In Milojicic *et al.* (2003), a list of large family of distributed systems can be found, which include, grid computing, cloud computing, peer-to-peer computing, cluster computing, client-server systems, and utility computing. Peer-to-peer network based technologies allow a distributed community of users to share a variety of digital or computer processing resources (Kumar *et al.*, 2011; Luther *et al.*, 2005). The difference of peer-to-peer networks with respect to traditional client–server networks is that they do not necessarily require centralized or dedicated servers (Krishnan *et al.*, 2003). Every node, or peer, that forms a part of the network may potentially contribute resources to other peers. As a result, peer-to-peer networks have many advantages over centralized networks, such as inherent scalability, no single point of failure, and self-administration capabilities (Li *et al.*, 2002). Fig. 2.3 shows an example of a peer-to-peer network that can be used for sharing computer resources. Any peer can communicate with other nodes in the network. A task that is required by one peer can be potentially allocated to other nodes, thereby allowing the peers to pool their resources.

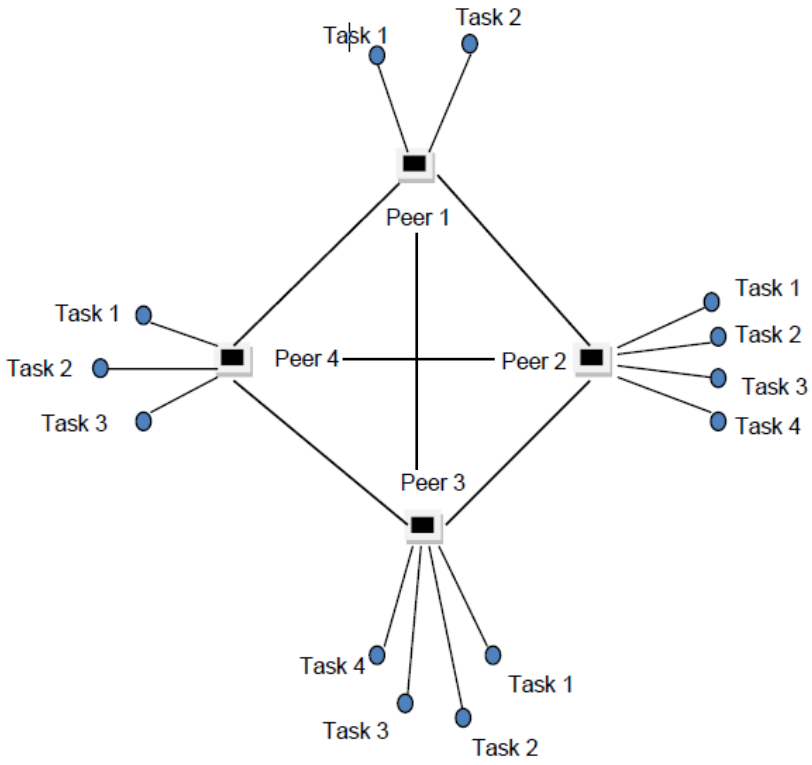


Fig. 2.3: A Peer-to-Peer Network for Sharing Computer Resources (Kumar *et al.*, 2011)

Fig. 2.4 illustrates another typical example of distributed computing system with the case of Cluster Computing scenario. Cluster computing has most often been the paradigm of choice for executing large-scale science, engineering, and commercial applications prior to the advent of grid computing. This is due to their low cost, high performance, availability of off-the-shelf hardware components and freely accessible software tools that that can be used for developing cluster applications. This is evident by the fact that the performance of cluster components has almost reached the performance of those used in supercomputers or HPC systems, in addition, the commodity components are improving in terms of performance and functionality all the time. The performance of workstation is doubling every 18 to 24 months (Hai *et al.*, 2001; Plaza *et al.*, 2009). The performance of network hardware and software is

improving with ever increasing bandwidths and later latencies being achieved between cluster nodes. However, they consume a lot of energy and this is another current issue that needs to be addressed by the scheduling community (Da Costa *et al.,* 2010).

As a simplified example, a cluster can be perceived in two different ways: either as a system connected to other networks and with specific needs in terms of access, or as a combination of heterogeneous systems (login nodes, computing nodes, storage nodes and so on) (Xue *et al*., 2012). Figure 2.5 illustrates the graphical representation of high performance computing cluster.

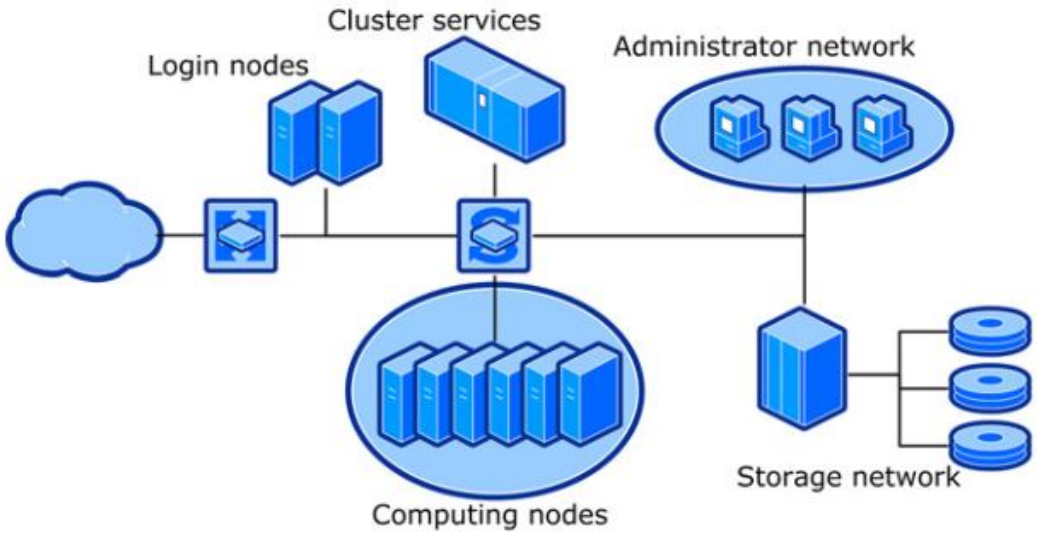


Fig. 2.4: Typical High Performance Cluster (Blanc and Lalande, 2013)

Challenges of distributed Systems

This section illustrates some of the main challenges of distributed systems and issues that tend to arise in their design (Nadiminti *et al*., 2006; Hooda and Bhadauria, 2013):

a. *Resource sharing*: this offers the opportunity to use available hardware, software or data anywhere in the system made available by the resource managers (Foster *et al.*,

2008; Foster, 2005). The resources managers control access, offer a scheme for naming, and control concurrency. A resource manager is a software module that manages resources of a particular type defined based on a particular resource sharing model. Resource sharing model describes how resources are made available, how resources can be used and how service providers and users interact with each other. Each of the aforementioned distributed computing systems type has different resource sharing model that differs from every other ones. For instance, in Object-based resource sharing model, any entity in a process is modeled as an object with a message based interface that provides access to its operations. More so, any shared resource is modeled as an object. Object based middleware such as CORBA, JAVA RMI defines object based resource model.

b. *Openness*: The openness of a computer system is the characteristic that determines whether the system can be extended and re-implemented in various ways (Lee, 2008). The openness of distributed systems is determined primarily by the degree to which new resource-sharing services can be added and be made available for use by a variety of client programs. In stronger terms, openness can only be achieved by making the specifications and documentations of the key software interfaces available to software developers. In other words, this requires publication of component interfaces and standards protocols for accessing interfaces. Systems that are designed to support resource sharing in this way are termed open distributed systems, to emphasize the fact that they are extensible. They may be extended at the hardware level by the addition of computers to the network and at the software level by the introduction of new services and the reimplementation of old ones, enabling application programs to share resources (Blair *et al.*, 2009; Hadim and Mohamed,

2006). A further benefit that is often cited for open systems is their independence from individual vendors.

c. *Scalability*: ability to serve more users, provide acceptable response times with increased amount of data. A system is scalable if it remains effective when there is a significant increase in the amount of work they need to perform (data) and number of users (Weil *et al.*, 2006). The designs of scalable systems present some challenges that revolve around: controlling the costs of physical resources, controlling the performance losses, and preventing software resources from running out and avoiding performance bottlenecks (Vaquero *et al.*, 2008). However, this is sometimes difficult to achieve, but there are methods which have proved advantageous such as the use of replicated data, the associated techniques of caching and the deployment of multiple servers to handle commonly performed tasks, enabling several similar tasks to be performed concurrently.

d. *Fault-tolerance*: maintain availability even when individual components fail (Dean and Ghemawat, 2008). When fault occurs in hardware or software, the active program may produce incorrect results or may even halt execution before they have completed the intended execution of the running program. Failures in a distributed system are partial and the implication is that, some components fail while others continue to function (Iamnitchi and Foster, 2000). Therefore the handling of failures is particularly difficult. There are quite a number of techniques available that can be used to deal with failures in distributed systems among which are; detecting failures, masking failures, tolerating failures, recovering from failures, and redundancy. Detail of the adopted techniques will come later in section 4.3.3 as progress is made in the design and implementation of the proposed work.

e. *Heterogeneity*: executing multiple tasks across network of systems irrespective of differences in network and hardware, operating system, programming languages, and implementation by different developers (Massie *et al.*, 2004). The Internet as an example of a distributed system enables users to access services, and run applications over a heterogeneous collection of computers and networks. The differences in the heterogeneity are masked by the fact that all of the computers attached to the different networks use the Internet protocols to communicate with one another.

2.2.2 Distributed Scheduling System

Scheduling is regarded very important to the success of distributed systems. Even the most powerful high performance computing environments require proper scheduling, in order to efficiently help in achieving both the user and resource providers' goals. For this reason, many researchers have been seeking scheduling methods to optimize distributed system performance. Furthermore, recent developments in computing environments, such as cloud, grid and heterogeneous clusters, present new challenges to schedulers. The heterogeneity of these systems and changes in their execution environment, often require dedicated scheduling techniques for each group of resources in the distributed computing environment. Distributed scheduling however, involves scheduling of tasks or computing resources in parallel alongside having the knowledge of the scheduling components to be distributed.

Scheduling of tasks on multiple computer nodes is addressed in Celaya (2012). This work addresses the problem of scheduling many tasks in environment of millions of unreliable nodes using the concept of decentralized or distributed scheduling model. The model utilizes the core advantages of a hierarchical network overlay, which

supports a scalable resource discovery and allocation scheme. It uses aggregated information to route tasks to the most suitable execution nodes, and is easily extensible to provide very different scheduling policies. The recorded achievement from this work include the implementation of policy that just allocates tasks to idle nodes, a policy that minimizes the global makespan and a policy that fulfils deadline requirements.

Karatza (2004a) considered the problem of scheduling in distributed systems. In this work, several scheduling policies employed in distributed server environments were presented, each of which addresses different aspects of the scheduling problem. The goal of this paper is to summarize research on distributed systems scheduling that is being performed worldwide. In addition Karatza (2004b) also studied scheduling in a distributed system with a time varying workload. This paper considers time varying distributions for job inter-arrival times, the parallelism of jobs and task service demand. These distributions represent real parallel system workloads. The weights of different workload parameters on performance metrics is examined to identify conditions that produce good overall system performance, and maintain fairness of individual job execution times. Simulation generates results that are used to compare different scheduling configurations. Four known scheduling policies are analysed and compared in this paper. The *First-Come-First-Serve* (FCFS), the *Shortest-Task-First* (STF), *the Limited STF* (LSTF-l, where the STF method is applied l time and then the oldest task in the queue is scheduled), and the *Job with the Smallest Number of Incomplete Tasks First* (JSNITF).

In Bahnasawy *et al.* (2011) a new task scheduling algorithm called Sorted Nodes in Leveled DAG Division (SNLDD) is introduced and developed for Heterogeneous Distributed Computing Systems (HeDCSs), with consideration for a bounded number of processors. The aim of the algorithm is to divide the Directed Acyclic Graph (DAG)

into levels, and sort the tasks in each level according to their computation sizes in descending order. The performance of the developed SNLDD algorithm against the Longest Dynamic Critical Path (LDCP) algorithm which is considered the most efficient existing algorithm shows that the proposed algorithm performed better in terms of speedup, efficiency, complexity, and quality.

Similarly, Long *et al.* (2011) proposed the development of an adaptive dynamic load balancing model in agent-based distributed simulations. A distributed approximate optimized scheduling algorithm with partial information (DAOSAPI), is presented in the work to tackle the problem of dynamic load balancing in the system. The proposed scheduling algorithm integrates the distributed mode, approximate optimization and agent set scheduling approach. Based on the experiments conducted to verify the efficiency of the proposed algorithm and the simulation performance under dynamic agent scheduling, the simulation results indicated that algorithm is robust and effective.

2.2.3 Distributed Artificial Intelligence and Multi-Agent System

The modern approach to artificial intelligence (AI) is centered on the concept of a *rational agent*. The term agent has various meaning in the literature. In this dissertation, the definitions given in (Chen and Tu, 2009; Zarandi and Ahmadpour, 2009) are adopted. An Agent is considered as a computing entity which has a definite purpose and can run in the distributed environment independently and persistently, and it generally has the following main characteristics: autonomy, reaction, interaction and initiative. This definition can further be extended by classifying agents as either weak or strong, thus, a weak agent is one that is (i) autonomous (ii) social (iii) reactive and (iv) pro-active. In addition, a 'strong' agent needs one or more of the following properties: (v)

mentalistic notions (beliefs, goals, plans, and intentions), (vi) rationality (vii) veracity and (viii) adaptability.

Agents are being used in a wide range of applications, including user interfaces (González, 2012), cloud information system (Yang, 2013), manufacturing (Darmoul *et al.*, 2013), network management (Alonso *et al*., 2013), workflow support (Corrêa da Silva *et al*., 2013) and data mining (Corrêa da Silva, 2013). While agents are widely used, a robust and maintainable architecture based on software engineering principles, such as object oriented design patterns has not yet been formulated (Kendall *et al.*, 1997). Software reusability, in particular, is still an important problem in agent based systems.

Agents are seldom stand-alone systems. In many situations they coexist and interact with other agents in several different ways. A typical example is software agents on the Internet. Such a system that consists of a group of agents that can potentially interact with each other is called a Multi-agent System (MAS), discussed in detail, in section 2.3, and the corresponding subfield of AI that deals with principles and design of multi-agent systems is called Distributed Artificial Intelligence (DAI).

DAI covers a wider range of issues that relates to distribution and coordination of knowledge and actions within the context of multi-agent environments. According to (Suárez Barón, 2011), there are number of reasons why DAI approach should be adopted. The first reason is, its application in developing large complex systems and the necessity to treat distributed knowledge in applications that are geographically separated, such as sensor networks, air-traffic control, or cooperation between robots. The second reason is to attempt to extend the cooperation between machine-machine or man-machine agents, and the third being the fact that DAI represents a new perspective

in knowledge representation and problem solving, due to the fact that it provides richer and realistic scientific applications.

2.2.4 Distributed Task Allocation, Coordination and Scheduling

Task allocation refers to the way tasks are selected, coordinated and assigned by scheduling agents to appropriate computational nodes. Within the context of distributed systems, and to be precise task allocation, coordination and scheduling, the following properties are considered as being distributed entities: computational resources (cores, memory, storage, bandwidths); information and knowledge of the tasks, agent capabilities, constraints, and the communication network topology; and the autonomous behaviour of agents to determine the task allocation and task coordination.

Jackson (2012) in his work stated that it is beneficial to exploit parallelism and distribute the computational burden across cooperating nodes. This is advantageous because for any distributed task allocation, coordination and scheduling the solution must be distributed (in some form) prior to execution. Communication between certain pairs of agents (including a possible central planner) may be sporadic, non-existent, unsecured, or delayed. Additionally, centralized adaptive behaviour for all task allocation mechanisms is an expensive approach, due to high computational costs, the cause of dimensionality and the single point of failure problem. For these reasons, distributed task allocation mechanism is considered for the proposed scheduling framework.

In multi-agents system and within the context of tasks allocation and scheduling, agents need to coordinate their behaviour in a decentralized manner in order to be able to solve complex allocation and scheduling problems and also achieve their design objectives (Mihail, 2012), aside that it is equally important that agents can communicate with each

other. Communication can be used to distribute the effort of determining the task allocation and task schedule across several agents. Candidate mechanisms for task allocation and scheduling are impacted differently, as a result of different assumptions on the topology of the underlying communication network. This dissertation considers some of these issues and provides algorithms to address the problem of a connected arbitrary network topology.

2.3 Multi-Agent System

A Multi-agent System (MAS) consists of a number of agents that work together to find answers to problems that are beyond the individual capabilities or knowledge of each entity (Long *et al.*, 2011). Multi-agent systems have been widely applied to analyse and solve complex problems, even in distributed scheduling systems (as discussed in Section 2.5.2). The interaction of agents can be designed such that the multi-agent system achieves a global goal. Although each agent is an intelligent entity, it is not enough for an individual agent to achieve the global goal in a distributed scheduling system. Multi-agent system approach can equally be used to model a robust and broad-spectrum scheduling system, in the form of a group of distributed, autonomous and adaptive intelligent agents. In this perspective, each agent has only a local view of the scheduling problem, but agents can cooperate with each other to perform tasks schedule, in parallel for the entire system through the autonomous interaction of the agents.

Multi-agent system comprises of several agents working in collaboration with each other. According to Graham (2001), agents are autonomous, heterogeneous, persistent, computing entities that have the ability to choose tasks on which to operate and when to perform them. Agents interact through communications with other agents to complete a task. In this context, a local agent may be acting on behalf itself, or an outside agent

might have contracted the local agent to complete a task. Wooldridge (2009) extended the concept of an agent to include an intelligent autonomous agent. The conceptual model is depicted in Fig. 2.5.

"*An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives*"

The concept of autonomy of an intelligent agent conveys that they are able to execute actions without intervention from humans or other systems. Again this concept is further characterised by the following: reactivity, pro-activeness and social ability (Logenthiran, 2012; Wooldridge, 2009).

a. *Autonomy*: this is the ability of agents to operate without the direct intervention of other agents or humans, and have some kind of control over their actions and internal state.
b. *Reactivity*: this refers to the ability of an agent to perceive, and understand any changes in its immediate environment, and respond in a timely fashion to changes that occur in it.
c. *Pro-activeness*: agents do not simply act in response to their environment, but they exhibit goal-directed behaviour by taking initiative.
d. *Social ability*: this is the ability of agents to interact with other agents (and possibly humans) via some kind of agent-communication language.

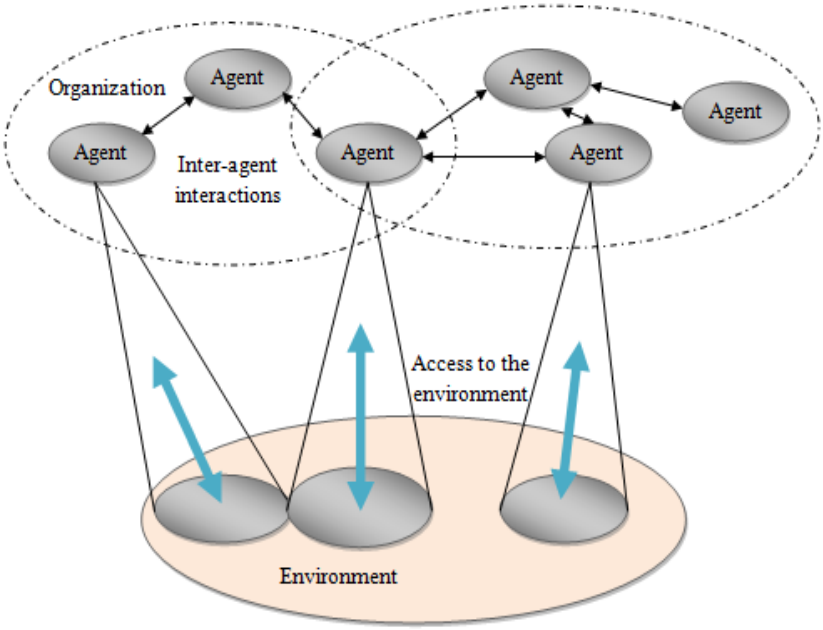


Fig. 2.5: Multi-Agent Systems as Computational Organisations (Jennings, 2001)

In the next subsections, a review of some of the major characteristics and advantages of the multi-agent system as it relates to the proposed study is discussed.

2.3.1 Characteristics of Multi-Agent System

Some of the main characteristics of intelligent agents are explained here to show the potential of the multi-agent system approach for building robust, scalable and cooperative distributed tasks scheduling system, that is adaptive in distributed computing environments. In Logenthiran (2010), some of the main characteristics of multi-agent system were mentioned with regard to its application in the control and management of distributed power systems. Similarly, in the work of Vlassis (2007) some of these characteristics were discussed at length.

a. *Autonomy*: Agents have a certain degree of autonomy. Agents can take decisions driven by a set of tendencies without a central controller. The autonomy of each agent is related to its resources.

b. *Environment*: Agents are capable of acting in an environment that is either static (time-invariant) or dynamic (non-stationary). This means that agents are capable of changing their environment through their actions.

c. *Communication*: In a typical dynamic multi-age scenario, agents interact amongst themselves, and this kind of interaction is associated with some form of communication. Basically, communication in multi-agent is viewed as a two-way process, where all agents can potentially be senders or receivers. Communication can be used in several cases, for instance, for coordination among cooperative agents or for negotiation among self-interested agents. Moreover, communication additionally raises the issues of what network protocols to use in order for the exchanged information to arrive safely and timely, and what language the agents must speak in order to understand each other (especially if they are heterogeneous). The proposed work offers to address this communication issue.

d. *Control*: Contrary to single-agent systems, the control in multi-agent system is typically distributed (decentralized). This means that there is no central process that collects information from each agent and then decides what action each agent should take. The decision making of each agent lies to a large extent within the agent itself.

e. *Knowledge*: In single-agent systems, it is typically assumed that the agent knows its own actions but not necessarily how the world is affected by its actions. In a multi-agent system, the levels of knowledge of each agent about the current world state can differ substantially. For example, in a team multi-agent system involving two homogeneous agents, each agent may know the available action set of the other agent, both agents may know (by communication) their current perceptions, or they can infer the intentions of each other based on some shared prior knowledge. On the other hand, an agent that observes an adversarial team of agents will typically be

unaware of their action sets and their current perceptions, and might also be unable to infer their plans. In general, each agent in a multi-agent system must also consider the knowledge of each other agent in its decision making. A crucial concept here is that of common knowledge, according to which every agent knows a fact, every agent knows that every other agent knows this fact.

f. *Agent Behaviour*: Agents have certain behaviours and tend to satisfy their objectives using their resources, skills and services. The way that an agent uses its resources, skills and services is defined by its behaviours. As a consequence, the behaviours of each agent are formed by its goals. For instance, an agent that controls a storage system aiming to provide uninterruptible supply to the load has different behaviour from that of another storage system in the system.

2.3.2 Agent Interaction and Communication

Agents do not pursue their individual goals alone, but in relationship with other agents. According to Salamon (2011) agents' interactions can be divided into three categories namely: coordination, cooperation and competition.

a. *Coordination* is a very efficient principle of conflict avoidance, because there is no need for negotiation, communication, or other kind of relationship between agents. To prevent further conflicts, agents are designed to obey to certain norms and rules.
b. *Cooperation* is the mode of interaction where a shared resource is presented, and the conflict avoidance cannot be resolved by rules. In this scenario, agents negotiate between them and work together to reach a common goal. The negotiation protocol should follow certain principles, such as efficiency, credibility, neutrality, distribution, simplicity and guarantee ending.

c. *Competition* between agents is unavoidable when negotiation is impossible or inefficient. Competition can be defined as the mode of iteration that occurs when the agents struggle for a scarce resource among themselves.

Communication and communication protocols are still being studied in terms of the efficiency of delivering their content as fast and as reliable as possible (Edelkamp *et al.*, 2004; Su *et al.*, 2008; Wooldridge, 2009). Much formalism has been developed for representing the properties of the communication. Nonetheless, it can be said that communication among cooperating agents usually happens through messages (Salamon, 2011). For the sake of clarity, agents do not communicate directly with each other, rather they use a messaging subsystem, which is responsible for receiving and delivering messages to the recipient.

2.3.3 Advantages of Multi-Agent Systems

The MAS approach has several distinct advantages over the traditional approaches for developing distributed tasks scheduling systems. Part of this merit is listed in (Logenthiran *et al.*, 2011; Sycara, 1998). MAS enhance the overall system performance specifically along the dimensions of computational efficiency, reliability, scalability, maintainability, flexibility and robustness. Some of the important advantages of the multi-agent system approach are briefly explained as follows;

a. Speedup and efficiency: due to the asynchronous and parallel computation.
b. Robustness and reliability: in the sense that the whole system can undergo a `graceful degradation' when one or more agents fail.
c. Scalability and flexibility: since it is easy to add new agents to the system.
d. Cost: assuming that an agent is a low-cost unit compared to the whole system.

e. Development and reusability: since it is easier to develop and maintain modular software than a monolithic one.

Multi-agent technology is a potential method that can be used to realize the benefits listed in (a) to (e), and most importantly, it can be used to model distributed scheduling system as is expected in our case, and most specifically to implement distributed tasks coordination among scheduling agents. However, introducing more agents also introduce complexity in the communication layer, since as the network grows it will take more time to find peers. To handle this complexity, MAS can use both centralized communication system (message queues), and decentralized one like gossip algorithms, which is best depends on the goal and possible size of the MAS.

2.3.4 Design Method for Multi-Agent System

The proposed conceptual design framework presented in Chapter three, is a suite of integrated method originating from existing technologies. These existing technologies include object-oriented methodology, design patterns, and software architecture in software engineering. In this Section, a review of some of the recent design methodologies that will guide the design of the proposed framework discussed in chapter three is presented.

Lynch and Rajendran (2004) stated that agent-oriented software engineering introduces a new level referred to as the *agent level*. The function of the agent level is to allow the software architect model a system in terms of interacting agents. This agent level offers greater abstraction than the object level but can build on and complement it. The authors argued that agent level has similarities with the object level: since it considers discrete entities interacting through message passing. However, agents tend to offer larger scale functionality than objects, responding to task-level service requests and often

communicating at a level where messages request agents to perform whole tasks or sub-tasks. In some cases, agents can be considered as experts albeit operating in highly restricted domains.

Reusability can also be considered as an important feature of the agent-oriented technology. Agent-oriented technology offers re-use in the form of re-useable agents and also in the form of stereotypical patterns of inter-agent communication. Additionally, small collections of agents can be re-used when these collections are assembled in standard ways, to perform specific high level tasks, these collections are described by some authors as holons (Bussmann, 1998).

An evolving standard for a design methodology to support multi-agent system is called Agent Unified Modeling Language (AUML), based on the Unified Modeling Language (UML) methodology which is used together with the object oriented systems. Bauer *et al.* (2001) identified UML as an appropriate basis for an agent design method. Odell explains that new technology is best accepted by industry if it is related to existing methods or, in this case, seen to extend existing methods (Odell *et al.,* 2000). On the user side, this minimizes the learning curve associated with adopting a new technique and reduces the risk that it will suffer from premature obsolescence.

Another major reason for basing agent design on object-oriented design methodologies relates to similarities that exist between agents and objects at some levels (Iglesias, 1999). However, at other levels differences between agents and objects are substantial and need new modeling constructs. Several authors have indicated UML to be insufficient for modeling agent-based systems and have suggested extensions and customizations, e.g. (Bauer, 2001; Kavi *et al.,* 2003; Bergenti and Poggi, 2000; Yim *et*

al., 2000). AUML has been proposed in an attempt to standardise extensions to UML (Odell, 2000).

The AUML community has proposed extending UML by introducing (i) Architecture Diagrams and (ii) Protocol Diagrams. Although, these new diagrammatic techniques are beneficial in specifying agent behaviour, there are still inadequacies. Additionally, it is believed that the amounts of additional notation that must be added to existing diagrams, in order to cover these inadequacies complicate the diagrams unnecessarily, and this in turn presents an additional barrier to multi-agent system novices. Two additional diagrammatic techniques are proposed for the purpose of this work, one which is effectively a UML collaboration diagram and the other which explicitly captures features like agent synchronization and concurrency.

2.3.5 Multi-Agent-Based Scheduling System Architecture

Multi-agent scheduling system architectures provide the frameworks within which agents are designed and constructed purposely to render scheduling services that address either specific or multi-dimensional scheduling problems (a scheduling strategy that would enable jobs with multiple resource requirements to be run effectively on a Grid computing environment), (Khoo *et al.,* 2007). According to Shen (2002), architectures for agent-based scheduling systems can be grouped into three categories namely: hierarchical, federated, and autonomous agent (Fig. 2.6 - 2.8d).

Hierarchical multi-agent based architecture: A typical distributed computing environment often comprises of a number of physically distributed, semiautonomous or autonomous processing units, each having partial control over local resources and with different information requirements. In this situation, a number of practical agent-based applications still use the hierarchical architecture, although critics often complain about

its centralized behaviour. In fact, agent-based distributed scheduling systems using functional decomposition usually have a hierarchical architecture because each agent type represents a specific processing function or a computing unit in a classical monolithic scheduling system.

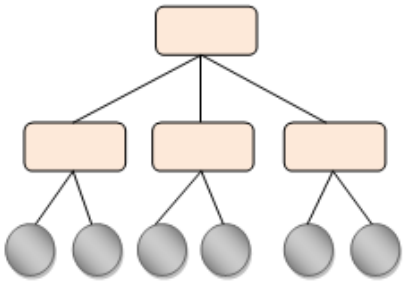


Fig. 2.6: Hierarchical Multi-Agent Based Scheduling Architecture (Shen, 2002)

Federated multi-agent based architecture: Due to limitations associated with the hierarchical architectures based on its centralized model, federated multi-agent architectures increasingly are considered a compromise solution for industrial agent-based applications, especially for large-scale engineering applications. A fully federated agent-based system has no explicit shared facility for storing active data. Rather the system stores all data in local databases and handles updates and changes through message passing.

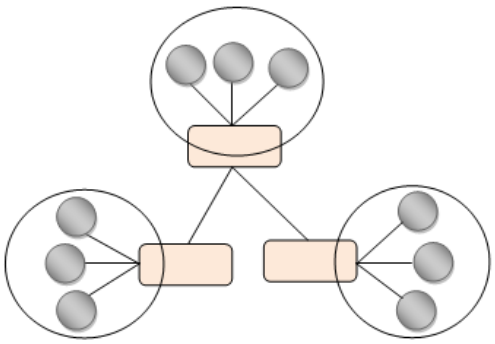


Fig. 2.7: Federated Multi-Agent Based Scheduling Architecture (Shen, 2002)

For agent-based distributed computing systems, four types of federated architectures can be mentioned: *facilitators*, *brokers*, *matchmakers*, and *mediators*. Facilitators (Fig. 2.7a) combine several related agents into a group. Communication between agents takes place through an interface, also called a facilitator. Each facilitator provides an intermediary between a local collection of agents and remote agents. It usually does this by providing two main services: routing outgoing messages to the appropriate destinations and translating incoming messages for consumption by its agents. Many agent-based collaborative design systems have used this approach (Shen and Norrie, 2007).

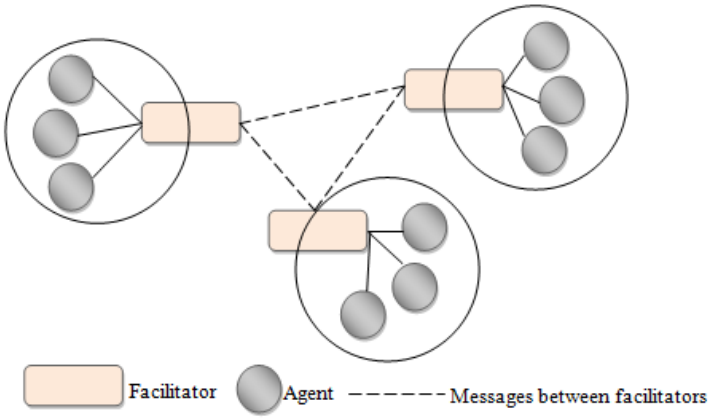


Fig. 2.7a: Facilitator Multi-Agent Architecture

Brokers (Fig. 2.7b) also called broker agents are similar to facilitators, with additional functions such as monitoring and notification. The functional difference between a facilitator and a broker is that a facilitator is responsible for only a designated group of agents, whereas any agent can contact any broker in the same system for finding service agents to complete a special task.

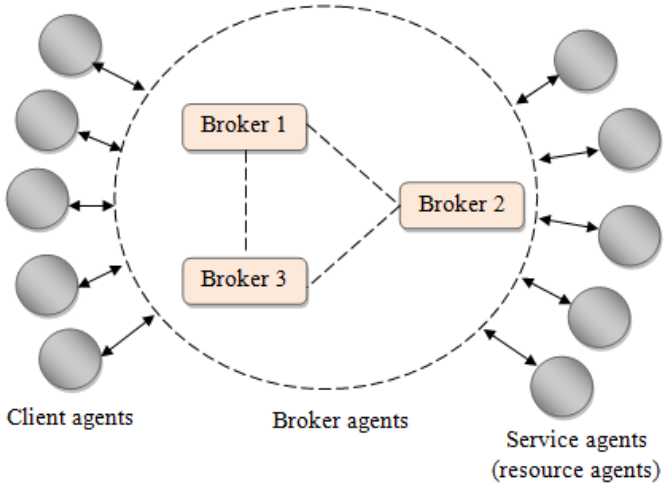


Fig. 2.7b: Broker Multi-Agent Architecture

The matchmaker architecture (Fig. 2.7c) is a superset of the broker architecture, because it uses the brokering mechanism to match agents. It is also referred to as the matchmaker agents, it mediate among both requester agents and provider agents for some mutually beneficial cooperation (Sycara *et al.*, 2002). Each provider agent must first register itself with a matchmaker. Provider agents advertise their capabilities by sending some appropriate messages describing the kind of service they offer. Every request a matchmaker receives will be matched with his actual set of advertisements. If the match is successful, the matchmaker returns a ranked set of appropriate provider agents and the relevant advertisements to the requester.

In contrast to a broker agent, a matchmaker does not deal with the task of contacting the relevant providers, transmitting the service request to the service provider and communicating the results to the requester. The reason behind this is to avoid data transmission bottlenecks, although this step might increase the amount of interactions among agents.

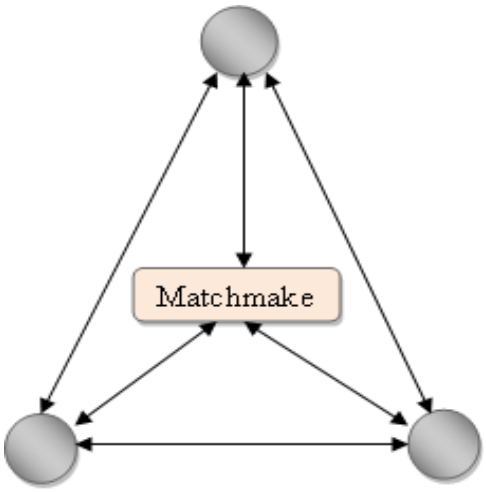


Fig. 2.7c: Matchmaker Multi-Agent Architecture

The *mediator architecture* draws from the approach used in the blackboard system. Besides functioning as a facilitator and a matchmaker, mediators assume the role of system coordinator by promoting cooperation among intelligent agents and learning from the agents' behaviours (Ebrahimi *et al.*, 2004). This architecture imposes a static or dynamic hierarchy for every specific task, which provides computational simplicity and manageability. It is suitable for developing distributed scheduling systems, that are complex, dynamic, and often comprises of many resource agents. Shown in Fig. 2.7d is the collaborative behaviour of the mediator agent which entails brokering and recruiting mechanisms.

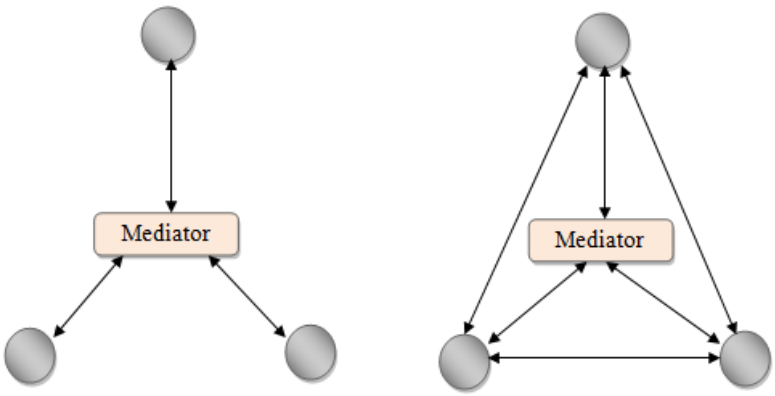


Fig. 2.7d: Mediator Multi-Agent Architecture

Autonomous multi-agent based architecture: Different definitions of autonomous agents exist, part of which have been defined in the previous subsection; however, an autonomous agent usually: i) is not controlled or managed by any other software agent or human being, ii) can communicate and interact directly with other agents in the system and with other external systems, iii) has knowledge about other agents and its environment, and iv) has its own goals and an associated set of motivations. The autonomous agent architecture is well suited for developing distributed scheduling systems consisting of a small number of agents (Fig. 2.8).

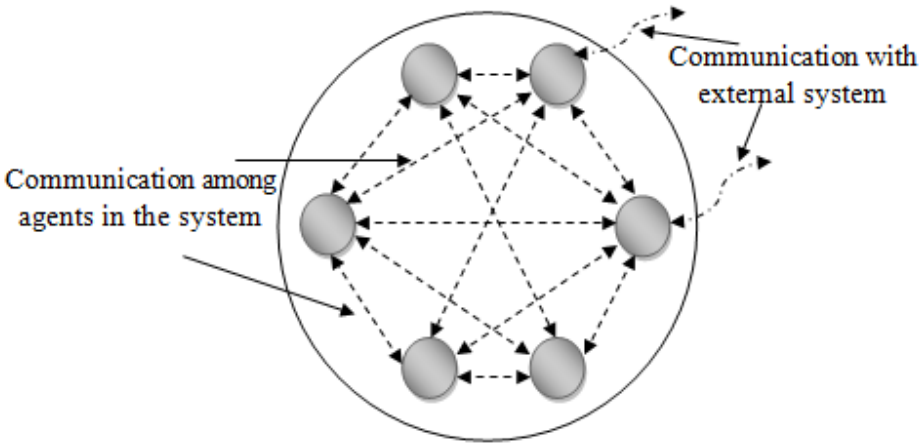


Fig. 2.8: Autonomous Multi-Agent Based Scheduling Architecture (Shen, 2002)

Autonomous multi-agent architectures can coordinate multi-agent activity through facilitation or mediation to reduce overhead, ensure stability, and provide scalability. They promise to be a good foundation on which to develop open, scalable multi-agent systems. One major advantage of autonomous multi-agent architectures is that many heterogeneous intelligent agents can be associated in a large distributed multi-agent organisation, that includes high-level facilitator agents.

2.3.6 Challenging Issues with Multi-Agent Systems

The multi-agent system technology offer many potential benefits but also raises challenging issues. Some of which are (Vlassis, 2007):

a. How to decompose a problem, allocate subtasks to agents, and synthesize partial results.
b. How to handle the distributed perceptual information. How to enable agents to maintain consistent shared models of the world.
c. How to implement decentralized control and build efficient coordination mechanisms among agents.
d. How to design efficient multi-agent planning and learning algorithms.
e. How to represent knowledge. How to enable agents to reason about the actions, plans, and knowledge of other agents.
f. How to enable agents to communicate. What communication languages and protocols to use. What, when, and with whom should an agent communicate.
g. How to enable agents to negotiate and resolve conflicts.
h. How to enable agents to form organisational structures like teams or coalitions. How to assign roles to agents.
i. How to ensure coherent and stable system behaviour.

The aforementioned issues associated with multi-agent technologies are interdependent, and their solutions may somehow affect each other. For example, a distributed planning algorithm may require a particular coordination mechanism; learning can be guided by the organisational structure of the agents. As part of the proposed work general goal, some of the issues raised in items (a) to (i) will be addressed.

2.4 Object-Oriented Software Design

The proposed framework integrates the object-oriented model and multi-agent system architecture, to explore the development of broad-spectrum scheduling system requiring the interplay of physical and software elements concurrently. Many classes of scheduling model could benefit from this approach. This section presents some of the related works on object-oriented software design. This section also covers some relevant preliminary conceptual design perspectives, which will aid in the design of the proposed general-purpose scheduling framework.

Object-oriented software design is a design strategy where system designers think in terms of 'things', instead of operations or functions. The executing system is made up of interacting objects that maintain their own local state and provide operations on that state information (Fig. 2.9). They hide information about the representation of the state and hence, limit access to it. An object-oriented design process involves designing the object classes and the relationships between these classes. When the design is realized as an executing program, the required objects are created dynamically using the class definitions.

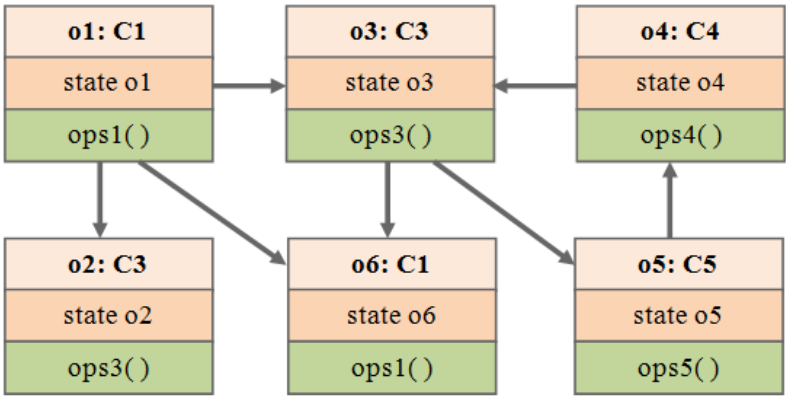


Fig. 2.9: System with Interacting Objects (Mishra *et al.*, 2007)

In addition, this technology offers a new and powerful model for writing software. OOP (Object-oriented programming) allows decomposition of a problem into a number of objects and then builds data and functions around these objects (Lejeune *et al.*, 2012). Specifically, object-oriented analysis and design is more cost-effective and a faster way to develop software and systems. This technology cuts development time, overhead and enables software engineers to make reusable, reliable and easily maintainable applications. The main goal of this research is to follow the principle of object-oriented software design approach in developing a general-purpose scheduling system. The object-oriented design model has for long been best choice for accomplishing the implementation of varieties of robust and scalable software, especially in the domain of sciences and engineering. For more detail on the applications developed using the object-oriented design approach (Wu *et al.*, 2012; Lejeune *et al.*, 2012; Anagnostopoulos and Burlando, 2012; Patzák and Rypl, 2012; Patzák, 2012; David *et al*., 2013; Martínez *et al.*, 2013).

2.4.1 Object-Oriented Database Design

An object-oriented database exposes the means through which objects can be stored and queried, using the same model that is employed by the application's programming language. In other words, an object-oriented database management system (OODBMS) extends the programming language with transparently persistent data, concurrency control, data recovery, associative queries, and some other capabilities (Rădulescu, 2009). The object-oriented databases have strong bindings to most object-oriented programming languages and platforms, including C++, Java, .NET, Perl, Python, Objective-C, and Visual Basic.

Unlike the traditional relational databases that have clear specifications, the object-oriented databases do not have any clear defined specification, but rather they depend on the database vendor objectives and the target object-oriented programming languages that OODBMS will bind to. However, some similarities still among the OODBMS products, most of which relates to the core object-oriented programming principles and foundations. The mandatory features depicted in Fig. 2.10 which must be present if a system is to be considered an object-oriented database are defined next:

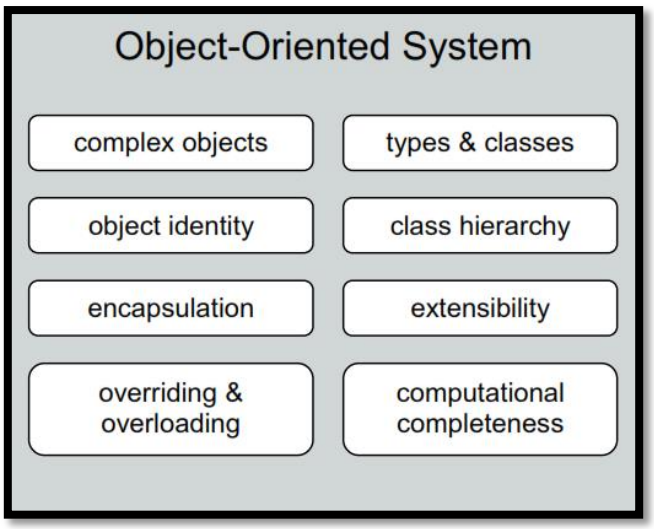


Fig. 2.10: Object-Oriented Database Features (Rădulescu, 2009)

Support for complex objects: A complex object mechanism allows an object to contain attributes that can themselves be objects. In other words, the schema of an object is not in first-normal-form. Examples of attributes that can comprise a complex object include lists, bags, and embedded objects (Maier, 2001).

Object identity: Every instance in the database has a unique identifier (OID), which is a property of an object that distinguishes it from all other objects and remains for the lifetime of the object. In object-oriented systems, an object has an existence (identity) independent of its value.

Encapsulation: Object-oriented models enforce encapsulation and information hiding. This means, the state of objects can be manipulated and read only by invoking operations that are specified within the type definition and made visible through the public clause. In an object-oriented database system, encapsulation is achieved if only the operations are visible to the programmer and both the data and the implementation are hidden.

Support for types or classes

a. **Type**: in an object-oriented system, summarizes the common features of a set of objects with the same characteristics. In programming languages, types can be used at compilation time to check the correctness of programs.
b. **Class**: is similar to type but associated with run-time execution. The term class refers to a collection of all objects with the same internal structure (attributes) and methods. These objects are called instances of the class.
c. Both of these two features can be used to group similar objects together, but it is normal for a system to support either classes or types and not both.

Class or type hierarchies: Any subclass or subtype will inherit attributes and methods from its superclass or *supertype*.

Overriding, Overloading and Late Binding

a. Overloading: A class modifies an existing method, by using the same name, but with a different list, or type of parameters.
b. Overriding: The implementation of the operation will depend on the type of the object it is applied to.
c. Late binding: The implementation code cannot be referenced until run-time.

Computational completeness: SQL does not have the full power of a conventional programming language. Languages such as Pascal or C are said to be computationally complete because they can exploit the full capabilities of a computer. SQL is only relationally complete, that is, it has the full power of relational algebra. While any SQL code could be rewritten as a C++ program, not all C++ programs can be rewritten in SQL.

Object-oriented storage schedule modeling

In relation to a scheduling system, an object-oriented database is the storage medium through which the schedule objects are kept and manipulated. One major role that the object-oriented database plays is the hierarchical storage of schedule objects types definitions together with their corresponding methods as well. This is often represented as the object type library that is referenced, by the scheduling agents during scheduling activities.

Example: Object types and their corresponding instances can be defined as follows: consider an object type "task", where $task_1, \dots, task_n$ are instances of the object type. All the n instances have the same attributes and as such the same method thus applies to all the instances, as shown in Table 2.1.

Table 2.1: Task Data Type and Instance

Task Object	Task Instance
Id	id = 1
Name	name = cluster
Type	type = 2
Quantity	quantity = 200
Priority	priority = 5
Ready	ready = 20
Due	due = 300
Method	Method
Create	Create
Copy	Copy
Delete	Delete
Display	Display
Insert	Insert
Update	Update

Between object types, there exist two very important relationships, which are, the "is-a" relationship and the "has-a" relationship. The "is-a" relationship signifies a generalization and the two types have similar characteristics. Often, the two object types are referred to as a subtype and supertype. For example, a "machine" object type is a special kind of "resource" object type. A "tool" object type is a different type of machine object type. A "has-a" relationship is an aggregation relationship and one object type comprises of a number of other object types. A "cluster" object is composed of several machine objects. A "data centre" object comprises of a number of cluster objects. A "routing table" object has tasks object as well as machine objects.

Object types related by "is-a" or "has-a" relationship have similar characteristics that include attributes as well as method. In other words, all the attributes and methods of the *supertype* objects will appear and be used by the corresponding subtypes. For

example, a machine object has all the attributes of a resources object and may also have some additional attributes. In a similar way, a cluster object has all the attributes of a machine object and all the methods applicable to the machine object can be applied to the cluster object as well. This is often referred to as inheritance.

Another concept that is important in object-oriented database design is the concept of template. A template is an abstract description of a method for different object types. The actual action will be taken according to the object type applied to. For example, the pre-process duplicate method can be defined as follows:

construct (object type undecided)
{
create object1 of object type undecided
}

The action performed by this method depends on the object type. A template may not be valid for all object types and may have some restrictions. There may also be a different interpretation of the template dependent on the object type that it is applied to. For example, splitting a tree object and splitting a set object has different interpretations.

An object can be retrieved and its methods can be triggered through commands which are similar to SQL commands in relational databases. The format of such a command is defined as follows:

Type, Object, Method, Message (object, variable, constant, ...):

Message: this is a means by which objects communicate, and it is a request from one object to another to execute one of its method. For example

Task_object.insert ("id", 1, ...)

This is a request to execute the *insert* method on a *Task* object

Method: defines the behaviour of an object. Method can be used to, (i) change state by modifying its attribute value, (ii) to query the values of selected attributes.

2.5 Scheduling Methods

According to Zhang (2003), scheduling methods can be categorized based on a number of approaches, such as the Object-Oriented and Control method (OOC), the Expert System method (ES), the Operations Research method (OR), the Artificial Intelligence method (AI), the Knowledge-base method (KB), the Neural Network method (NN), and the Multi-agent method (MA). Each of the aforementioned scheduling methods alongside their strengths and weaknesses are explained subsequently.

2.5.1 Object-Oriented Method and Control (OOC)

This method came about as a result of trying to find a solution to build a scheduling system that is more generic in it application and also to provide a guideline that facilitates and standardizes the design and development of such scheduling system (Pinedo and Yen, 1997). In terms of this method, computing or IT infrastructure are viewed as a set of objects. The scheduling environment in this case consist of IT resources, which can be divided further into objects, classes and so on. Based on Object-Oriented Design (OOD) principles and concepts, the behaviour of these objects is described in terms of methods. Again, applying the same concept, the real world can be represented in a more realistic manner so as to manipulate a system abstraction in a more direct way.

Objects behaviours are widely represented using the object-oriented programming languages (OOP), such as C++ or Java. However, this method provides an

encapsulation mechanism for generic behavioural knowledge that is not explicit in the rest scheduling methods discussed, in the next subsections. The OOC method is flexible, re-usable and extensible; therefore, the various components of the scheduling environment can be analysed in a smaller entity, accurately simulated and readily controlled. The key factor to the OOC approach is how to use mechanisms for representing objects and sharing behaviours.

Gimenes *et al.* (2000), presented an object-oriented framework for task scheduling in domains such as process-centered software engineering environments and workflow management systems. Similarly, Patzák and Rypl (2012) presented a design and implementation of parallel load-balancing framework in an object-oriented finite element environment. The parallelization strategy is based on domain decomposition and message passing paradigms.

2.5.2 Multi-Agent Method (MA)

The multi-agent scheduling method has been of recent widely adopted, deployed and also has proven to be even more effective in its usage and operations especially in the areas of tasks scheduling and resource management systems (Wu *et al.*, 2011). Basically, the multi-agent scheduling approach proposes the utilization of cooperative distributed agents either from the same domain or different domain to play an independent role in scheduling activities for common goals. However, the development of flexible multi-agent is a key factor in adopting this method; otherwise, it offers better advantage for developing robust and intelligent scheduling systems. Suarez Baron (2011) discussed in detail the adoption of this scheduling method and its usage in developing a dynamic task-allocation scheduling system in multi-agent environment.

2.5.3 Operations Research Method (OR)

Scheduling systems based on operations research methodologies are characterised, by a reduction of the problem into a mathematical programming formulation, and subsequent solution by formal algorithm designs. This approach faces the difficulty that the problem quickly becomes computationally infeasible for even moderately sized problems, due to the large search space generated by the model. An additional disadvantage is that such model does not directly reflect the natural structure of the domain and hence makes it difficult to reduce the problem search space. In recent years however, some OR researchers have been advocating the use of simple heuristics rules to tackle large problems (Noronha and Sarma, 1991), but these lack the advantage of a unified approach and are seen as a poor relation to the formal method.

2.5.4 Knowledge-Base Method (KB)

The knowledge-base method for solving scheduling problems is an aspect of the artificial intelligence "AI" domain. Several knowledge-base approaches have emerged, using different paradigms, in particular, constraint-based search, heuristic search, and rule-based methods (Sauer and Bruns, 1991; Sauer, 2001; Abdullah *et al.*, 2006). Although, most of these approaches tend to address predictive scheduling problems, the KB approach concentrates majorly on knowledge acquisition, expression and application. In essence, it has to construct objectives, analyse physical objects and then find a scheduling solution by using knowledge. Knowledge on any object can be used, with the result that the scheduling solution should be more rational, as compared with other methods. The main problem with this approach lies on how to define knowledge and add knowledge to real-time objects scheduling solutions.

2.5.5 Rule-Based Approach

The rule-based scheduling approach is defined based on the assumption that behaviour can be described incrementally by adding "rules" to the behaviour until the correct behaviour is attained for the system. This approach has proven to be hugely successful for some problems of heuristic nature. The drawback of this approach has to do with the implicit description of sequencing of the scheduling structured elements or components, which obscures obvious time relation between events in tasks. This approach is better appreciated when combined with other scheduling methods, such as the object-oriented method for instance, so that the encapsulated knowledge is represented in rules, instead of in a traditional procedural language. Zhang (2003), described this hybrid approach of combining OO method and rule-based approach, while Frincu (2011) discussed the application of the rule-based approach in developing an adaptive scheduling for a distributed system.

2.5.6 Neural Networks Method (NN)

Neural networks (NN) are collections of mathematical models that emulate some of the observed properties of biological nervous systems and draw on the analogies of adaptive biological learning (Tang *et al*., 2005). The key element of the NN paradigm is the novel structure of the information processing system. The advantage of NN lies in their resilience against distortions in the input data and their capability of learning. These features give them a good perspective to the field of dynamic scheduling. The key research hinges upon how to structure neural network by traditional methods and strategies, and how to simulate a real-time environment. Chen *et al*. (2007) combined a competitive scheme with slack neuron into Hopfield neural networks to solve multiprocessor real-time job scheduling problems.

2.5.7 Expert Systems Method (ES)

The expert systems consist of a rule base, a snapshot of the current solution, and an inference engine. The inference engine determines how the if-then rules in the rule base are applied to the current solution, in order to execute the search. The rule base may be expanded as the solution progresses. The expert system method is highly problem oriented and is often deployed to tackle specific scheduling problems within a specific scheduling domain based on a specific knowledge (Zhang, 2003; Metaxiotis, *et al*., 2002). Its drawbacks lies on the fact that an independent procedure must be developed for the problem, and not for the general-purpose of scheduling. An artificial immune-system based scheme was proposed to solve the dynamic economic dispatch problem of engineering units (Basu, 2009b).

Notably, none of the aforementioned scheduling methods can be used to solve complex scheduling problems without the integration of one or more other methods. Therefore, there is need for an alternative scheduling solution that requires the combination of more than one of the above discussed scheduling methods in providing solutions to more complex scheduling problems. In this dissertation, a proposal of a hybrid scheduling system that consists of a mix of both the Object-oriented method and Multi-agent scheduling approach is made. This is in line with the targeted development of a broad-spectrum scheduling framework, specifically, designed for scheduling of multi-component applications using a multi-agent systems.

2.6 State of the Art Scheduling Systems

Since scheduling applications are diverse, people from different fields have studied them for decades. During the last six decades, many different techniques have been developed and applied to scheduling problems, including critical path methods, priority

rules, dispatching rules, dynamic programming, branch and bound. Unfortunately, most of the scheduling problems are classified as NP-hard in complexity theory (Brucker, 2007), and by implication it implies that in general, there is probably no polynomial-time algorithm for solving the problem, and therefore it is regarded as a computationally intractable problem. This however, contributed to the reason why many researchers were compelled to applying different combinations of scheduling techniques, to develop scheduling systems.

Analyzing a particular scheduling problem and applying a practical method to obtain an optimal or sub-optimal solution, is only a minor part in the solution of scheduling problem. Developing a scheduling system that can be integrated into the different classes of distributed system environments, and that will scale and perform efficiently, is even more difficult to achieve. In addition, each scheduling applications directed to these distributed computing environments like the clouds, grids, and peer-to-peer systems all have completely different implementation approach. Consequently, scheduling domain developers, often have to develop systems for each new application in each area of problem interest. This however, is not economical as it consumes valuable resources and time. Another approach will be to build a general-purpose scheduling system that will solve most scheduling problems especially for those environments that share similar characteristics.

In the subsequent sub-section, an overview of the main related works focused on the topics addressed in this dissertation is presented.

2.6.1 Scheduling Algorithms Overview

In Dong and Akl (2006) scheduling algorithms were classified into hierarchical taxonomy according to several criteria (Fig. 2.11).

A. **Local** vs **Global Scheduling**: by local scheduling policy, this refers to the procedures determining how the processes resident on a single CPU are allocated and executed. The global scheduling policy requires making use of the targeted system information to assign processes to multiple processors, so as to optimize a system-wide performance objective (Corman *et al.*, 2014; Gopalan and Chiueh, 2002; Davis and Burns, 2005; Kumar *et al.*, 2014)

B. **Static** vs. **Dynamic Scheduling:** this classification which falls under the global scheduling, divides scheduling algorithms based on when the scheduling decisions are made. In the case of static scheduling, information regarding all resources in the distributed system as well as all the tasks in an application is assumed to be available prior to the scheduling events. Whereas in the case of dynamic scheduling, the basic idea is to perform task allocation during application execution concurrently. This is useful especially when it is impossible to determine the execution time, direction of branches and number of iterations in a loop as well as in the case where jobs arrive in a real-time mode. This approach tends to put into consideration the dynamic nature of distributed systems, and is preferred to its static counterpart since within the context of dynamically distributed system environment, resources properties are bound to change during application task's execution (Lucas-Simarro *et al*., 2013; Hao *et al*., 2007; Yu *et al.*, 2008)

C. **Optimal** vs. **Sub-optimal**: this classification is based on the assumption that all information regarding the state of resources and the application jobs is known beforehand, in such case, an optimal assignment could be made based on some objective function, such as minimum *makespan* and maximum resource utilization. However, due to the NP-Complete nature of scheduling algorithms and the difficulty in distributed system scenarios to make reasonable assumptions which are

usually required to prove the optimality of an algorithm, current research tries to find suboptimal solutions by using scheduling heuristics or approximate algorithms. For optimal solutions, a very costly solution would be to use linear programming. This however grows exponentially as the number of tasks and resources grows (Verma and Kausha, 2014; Zhang *et al.*, 2014)

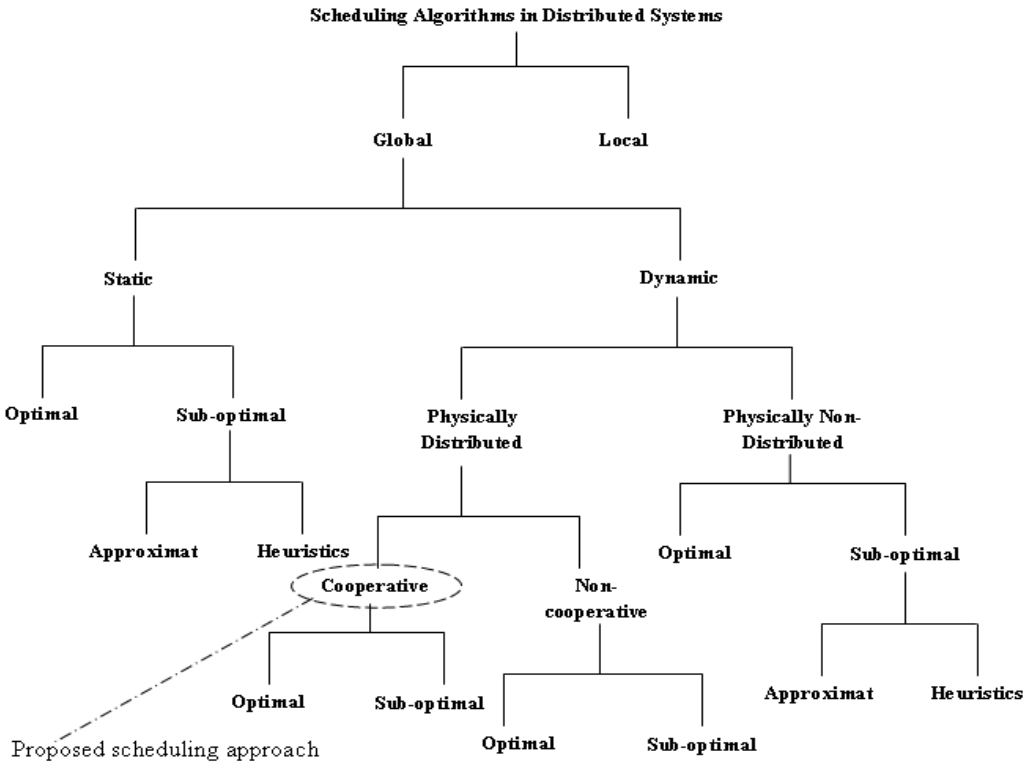


Fig. 2.11: A Hierarchical Taxonomy for Scheduling Algorithms (Dong and Akl, 2006)

D. **Approximate vs. Heuristics**: this classification groups scheduling algorithms based on the techniques of using computational model, for approximate search space and realistic assumptions. The approximate algorithms use formal computational models, but instead of searching the entire solution space for an optimal solution, they are satisfied when a solution that is sufficiently "good" is found. In the case where a metric is available for evaluating a solution, this technique can be used to decrease the time taken to find an acceptable schedule. The heuristic scheduling algorithms represents the class of

algorithms which make the most realistic assumptions about a priori knowledge concerning processes and system properties. It also offers an alternative solution to scheduling problems, however, it rarely provides optimal solutions, but only require the most reasonable amount of cost and system resource properties to operate. The evaluation of this kind of solution is usually based on experiments in the real world or by means of simulation (Gawiejnowicz and Kononov, 2014; Arnaout *et al*., 2014; Dalfard *et al.,* 2012).

E. **Distributed vs. Centralized**: the scheduling algorithms belonging to any of these two classes, rely on either making its scheduling decision on a single centralized scheduler or on multiple distributed scheduler (Fig. 2.12- 2.14). The centralized approach is easier to implement but has scalability issues and minimal fault tolerance. The decentralized scheduling algorithms on the other hand, are scalable and also have high fault-tolerance. Having multiple schedulers allow sub-parts of the system to work independently even when connection between peer schedulers fails. They will continue to function and schedule tasks in their own part of the Distributed System (DS). Results which need to be transmitted to resources inside the network area, where the connections have been interrupted will be resumed once broken connections are re-established. New incoming tasks are subject to the same treatment. Distributed schedulers can be viewed as either cooperative or non-cooperative depending on whether scheduling decisions are taken with regard to other schedulers or not. Christodoulopoulos *et al.* (2009) give an example of a cooperative distributed scheduler and compare the results with a centralized and local scheduling (Dalfard and Mohammadi, 2012; Augonnet *et al.,* 2011).

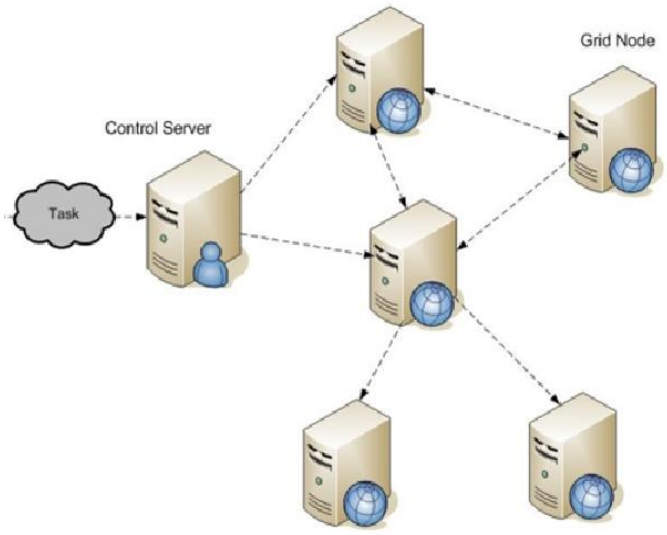


Fig. 2.12: Centralized Scheduling Model

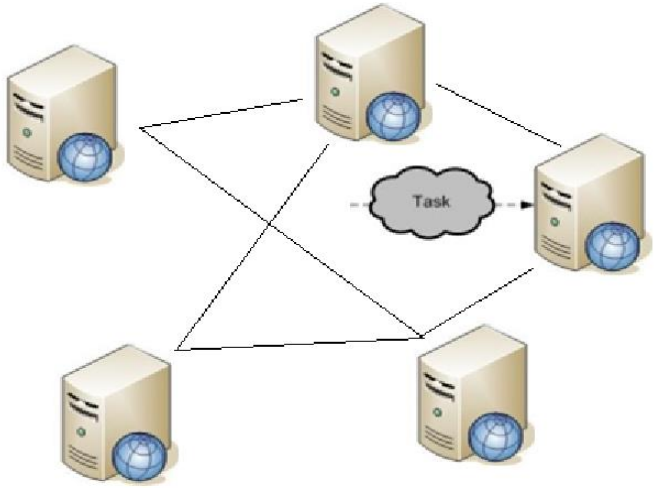


Fig. 2.13: Distributed Scheduling Models

F. **Cooperative vs. Non-cooperative**: This scheduling technique concerned with majorly distributed scheduling algorithms and its classification, is based upon whether the computing nodes involved in tasks scheduling are working cooperatively or independently. In the non-cooperative case, individual schedulers act alone as autonomous entities and arrive at decisions regarding their own optimum objects independent of the effects of the decision on the rest of system. In the case of the cooperative scheduling, each scheduler has the responsibility to carry

out its own portion of the scheduling task, but all schedulers are working towards a common system-wide goal. Each scheduler's local policy is concerned with making decisions by interacting with other schedulers in order to achieve some global goal, instead of making decisions which only affects local performance or performance of individual particular task. In Rahman *et al*. (2010), cooperative scheduling was presented and in Donassolo *et al.* (2011), non-cooperative scheduling was broadly covered.

2.6.2 Scheduling and Resource Management Systems Overview

In distributed system environment, scheduling algorithms play an important part in deciding when and where to perform scheduling functions for every incoming or already submitted task. This implies that the scheduling algorithm function is restricted to taking scheduling decision alone. In order to apply the scheduling decisions taken by a scheduling algorithm, a resource management system (RMS) is required. The RMS provides a set of services which vary depending on the system but usually involve taking tasks and assigning them to resources, based on the logical assignment done by the scheduling algorithm.

Distributed systems environment are characterised by a number of unpredictable fluctuations in resource and network bandwidths, yet they are both rigid and flexible in what concerns interoperability and interactivity. An excellent possible solution to these problems could be the adoption of agent based scheduling and resource management approaches. Agents are known for flexibility, agility and autonomy and as depicted in (Suarez Baron, 2011) they tend to adapt very well in such an environment that is challenging. More specifically, agents' environments are dynamic, unpredictable and

highly unreliable. This means that agents cannot assume that their environment will remain static while they try to achieve their targeted goals.

Despite the existence of numerous classical scheduling and resource management systems, the agent based approach has equally received much attention from researchers in recent times. Some of these recent developments related to the present work are presented below:

Li and Li (2012) proposed a model of an ad hoc grid resource management system, the producers and consumers of ad hoc grid resource are modeled as the self-interested decision-makers described in microeconomic theory. All market participants in the ad hoc grid environment including grid resources and services, could be represented as agents. An economic agent is applied to build an ad hoc grid resource management system, where ad hoc grid resource consumers and providers can buy and sell ad hoc grid resource, based on an underlying economic architecture. The main processes involved in ad hoc grid resource management are resource registration, discovery, and resource allocation. Comparisons of the proposed ad hoc grid resource allocation algorithm with other existing ad hoc grid resource allocation algorithm were made based on the simulated experiment performed. The simulation results showed that the proposed algorithm performed better than the other compared allocation policies. The proposed ad hoc grid task agents and ad hoc grid resource agents do not communicate directly with one another or amongst themselves. All interactions are by the means of ad hoc grid market which serves as centralized agents that perform the function of delegating tasks to other contributing agents. The implication is that collaborating agents are not aware of each other's capability and this tends to limit the overall performance of the system. Again, the centralized agent model (ad hoc grid market agent) does not offer good fault tolerance in case of any eventualities.

In the work presented in Cao *et al*. (2005), a Grid load balancing approach was implemented by combining both intelligent agents and multi-agent approaches. Each existing agent is responsible for handling a portion of the task scheduling event across multiple resources within a Grid. Inside the agent environment, there exists a hierarchy of agents which cooperate with each other in a peer to peer mechanism with the aim of achieving their common goal of finding new resources for their tasks. This hierarchy is composed of a broker, several coordinators and simple agents. By using evolutionary processes (Mihaila, 2011; Mihaila and Mihaili, 2008), the scheduling algorithms are able to cope with changes in the number of tasks or resources.

In the work of Wu *et al*. (2011), a novel multi-agent reinforcement learning method named Ordinal Sharing Learning (OSL) method, is presented for job scheduling problems in distributed system environment. This method has been proposed to resolve issues concerning load balancing in computational Grids. The approach presented, addresses scalability issue by using a routine-based distributed learning strategy, which achieves agents coordination through an information-sharing mechanism, with limited communication. Simulation results estimated that the OSL method can attain the goal of load balancing effectively, and its performance is even comparable to some centralized scheduling algorithm in most cases. The convergence property and adaptability of the proposed method were also explained in the work. However, even though the proposed model is assumed to have partially performed well on the aspect of load balancing, the challenges regarding inter-task synchronization, dynamic scheduling and robustness features were not considered in their method.

In the work of Shah *et al.* (2012a) architecture of an agent-based scheduling framework for the Grid computing is proposed. This framework is based on a hierarchical multi-layer and hierarchical agent-based architecture and is designed to offer a robustness

feature to a Grid scheduler. A robust Grid scheduler will compute and deliver jobs effectively and efficiently, despite the constant changeability of Grid computational environment, such as heterogeneity of jobs, resources and network conditions. In the work, high claim achievements were made without any experimental proof whatsoever, the challenges that come with the dynamic nature and heterogeneity of the distributed environment were not in any way addressed, and also the problems of domain adaptability and platform scalability were not equally captured by the proposed methods.

Shah *et al.* (2012b) presented a study on the design and evaluation of agent based prioritized dynamic round robin scheduling algorithm on computational grids. The proposed model introduces a concept of fairness to scheduling and presents a new agent based job scheduling algorithm called Agent based Prioritized Dynamic Round Robin (APDRR). The APDRR is designed and developed by combining the best features of round robin and priority job scheduling algorithm using agent technology. APDRR represents fair scheduling algorithm from a users' point of view while also regarding the optimization criteria that are anticipated from the systems' perspective. The authors claimed based on their experimental result obtained, that the proposed APDRR scheduling algorithm poses a high degree of optimality in performance, efficiency and scalability. One limitation of the proposed round robin agent-based algorithm is that its implementation is based on the master-slave architecture. The master-slave architecture is akin to a centralized scheduling system that is faced with the problems of fault tolerance, communication overhead and adaptability.

There are a number of accomplished non-agent resource management platforms that also pursue similar goals like the proposed research direction, such as the work of Hindman *et al.* (2011) that introduced a fine grained resource sharing platform for data

centre computing known as MESOS. The MESOS platform offers the sharing of commodity clusters between multiple diverse cluster computing frameworks like Hadoop and MPI. The system shares cluster resources in a fine-grained manner, thereby allowing different independent frameworks to achieve data locality by taking turns reading data stored on each local machines. MESOS provides support for distributed two-level scheduling mechanism called resource offer. This mechanism allows the system to decide the amount of resources to offer for each participating framework, while the frameworks decide which resources to accept and which computations to run on them. Some of the advantages that the platform provides are that, it can achieve a near-optimal data locality when sharing computing resources among diverse frameworks, its scalable and resilient to failure.

Hindman *et al.* (2009) presented an argument that no single framework is able to offer an optimal solution to all applications, and that organisations should instead be allowed to run multiple frameworks efficiently in the same distributed resource sharing environment like the cloud for instance. Their work further reinstated that, in order to ease the development of new frameworks, it would be critical to identify common abstractions and modularize their architectures. It was based on this perspective that they proposed the development of a low-level substrate that provides isolation and efficient resource sharing across frameworks running on the same clusters. The platform offers each framework the freedom to implement its own programming model and fully control the execution of its jobs. The proposed application referred to as NEXUS, tends to foster innovation in distributed resource sharing environment by letting organisations run new frameworks alongside existing ones by also allowing developers to concentrate on building domain specific applications rather than developing a general-purpose resource sharing frameworks.

2.7 Survey of Related Approaches and Contribution of Dissertation

In this section, a survey of the relevant literature in and about dynamic task allocation and coordination in multi-agent environments is carried out. To summarize this presentation, a summary of those gaps within the multi-agent distributed tasks scheduling literature which this dissertation is out to address is presented.

2.7.1 Adaptive Scheduling for Distributed Systems

More closely related to the proposed research area is the work of Frincu (2011). The dissertation addressed some of the main issues related with task scheduling inside distributed heterogeneous systems, among which include the ones caused by the influence of system volatility over the scheduling heuristics and the problems caused by failures in some of the scheduling platform components. The relationship between current (reviewed) and proposed work lies in the fact that the author also proposed the implementation of a self-healing multi-agent system for task scheduling.

The proposed inter-provider scheduling platform according to the author offers a fully distributed storage and communication mechanism, possess self-heal ability so as to support fault-tolerance by means of implementing agents as recoverable modules of feedback control loops, support autonomy among providers by separating the dynamically changing scheduling policy from the application, by means of an inference engine and the system, is also able to change the scheduling policy at runtime based on the current system configuration. The novelty of the platform is in its self-healing capability which was achieved by mixing a series of existing software solutions with software engineering concepts (e.g., multi-agent and rule-based method discussed earlier on in our current work). Two main aspects were considered essential in this

scope: distributiveness of the MAS components and the component monitoring and decision interface.

One major drawback with the work is that the multi-agent performance greatly depends on some sets of rules incorporated in the system inference engine. These rules involve implicit description of sequencing of the scheduling structured elements or components, which obscures obvious time relation between events in tasks. And again, the system requires the services of some external predictive heuristics algorithms that provide only estimated guesses for optimality. However, in this dissertation, a different hybrid scheduling approach, that leverages the intelligence of multi-agent systems and the effectiveness in following the object-oriented design method is proposed. In essence, this involves a mix of two scheduling methods, multi-agents and object-oriented scheduling methods, to address the proposed dissertation' objectives of designing a general purpose scheduling system, which is capable of addressing the research problem statement identified in section 1.3.

There are many advantages in following an object-oriented design approach in the development of scheduling system. First, the design is modular, which makes maintenance and modification of the system relatively easy. Second, large segments of the code are reusable. This implies that two scheduling systems that are substantially different still may share a significant amount of code. Third, the designer thinks in terms of the behaviour of objects, not in low-level detail. In other words, the object-oriented design approach can speed up the design process and separate the design process from the implementation process.

2.7.2 Multi-agent Cluster Scheduling for Scalability and Flexibility

The work presented in Konwinski (2012) describes a taxonomy and evaluation of three cluster scheduling architectures designed, for scalability and flexibility using a common high level taxonomy of cluster scheduling, Monte Carlo simulator, and real system implementation. The work first considered the popular Monolithic State Scheduling (MSS), and then describes the development of two new architectural models namely the Dynamically Partitioned State Scheduling (DPS) and Replicated State Scheduling (RSS). Evaluation of the DPS, which uses pessimistic concurrency control for cluster resource sharing, is made. Similarly, a design, implementation, and evaluation of MESOS, a real-world DPS cluster scheduler that allows diverse cluster computing frameworks to efficiently share resources was presented.

In their work, a shift from beyond monolithic state scheduling model into multi-agent scheduling approach was considered, so as to increase the scalability and flexibility of their proposed cluster scheduling, by allowing scheduling decisions to happen concurrently. This technique is made possible with the introduction of a different type of multi-agent scheduling technique called Partitioned State Scheduling (PSS), in which Cluster State is partitioned into non-overlapping scheduling domains. One advantage of this scheduling mechanism is that it allows multiple scheduling decisions to be made in parallel. However, the implementation of the PSS model is in line with Mesos architecture. Mesos implements a multi-agent scheduling model, where each framework decides which offers to accept from a centralized master agent.

There are three major limitations to the proposed partitioned state scheduling technique, first the problem of fragmentation: When tasks have heterogeneous resource demands, a distributed collection of frameworks may not be able to optimize bin-packing as well as

a centralized scheduler. Second, is with the issue of interdependent framework constraints: It is possible to construct scenarios where, because of esoteric interdependencies between frameworks (e.g., certain tasks from two frameworks cannot be co-located), only a single global allocation of the cluster performs well. And lastly, the problem of framework complexity: Using resource offers may make framework scheduling more complex. This is because the resource offer policy in mesos restricts frameworks to base their resource choice on what mesos offers them.

2.7.3 Decentralized Coordination in Multi-agent System

Mihail (2012), proposed a framework model for decentralized coordination in multi-agent systems. The research has been inspired mainly by the challenging domain of wireless sensor networks (WSNs). WSNs are an example of a decentralized system with complex objectives, but no central authority to compute a global solution. Agents (or sensor nodes) are fully cooperative, but also highly constrained with limited computational resources and restricted communication range. The WSN problem in particular, requires agents to synchronize with some nodes, in order to improve message throughput, and at the same time desynchronize with others, in order to reduce communication interference.

The main contribution of this work lies on the development of a simple decentralized reinforcement learning approach, called the Win-Stay Lose-probabilistic-Shift (WSLpS). This method allows highly constrained agents to efficiently coordinate their behaviours, by imposing minimal system requirements and overhead. The work demonstrates that it is possible for a global coordination to emerge from simple and local interactions without the need for a central control or any form of explicit coordination. Despite its simplicity, the proposed WSLpS model claims to quickly

achieve efficient collective behaviour both in pure coordination games and in pure anti-coordination games.

The author deployed the proposed approach through the design of an adaptive low-cost communication protocol, called DESYDE (DEcentralized SYnchornization and DEsynchronization communication protocol or learning algorithm), which according to them achieves efficient wake-up scheduling in wireless sensor networks. In this way, they were able to demonstrate how a simple and versatile approach can be used to achieve efficient decentralized coordination in real-world multi-agent systems. One major limitation with the proposed WSLpS is due to the lack of global knowledge among contributing agents, individual agents cannot determine whether a final solution has been reached or not.

2.7.4 Distributed Problem Solving by Means of Agent Negotiation

Distributed problem solving usually refers to such an environment where planning, solving, or coordination activity is distributed across multiple actors, processes, or sites. Distributed planning has been viewed as either (a) planning for activities and resources allocated among distributed agents, (b) parallel computing aimed at plan construction or (c) plan merging activity (Vokrinek *et al.*, 2011). In this work, an abstract architecture of a multi-agent solver and respective algorithms that provides decomposition, task allocation and task delegation, with the intention of implementing a wide variety of practical multi-agent planning and problem solving systems was presented.

The architecture is based on a social welfare maximization using agent negotiation over task allocation, delegation and reallocation. The proposed interaction mechanism ensures global optimization behaviour while the individual agents apply local planning heuristics and strategies. The process of coordination and interaction between agents

depends strongly on the ability of individual agent actors to perform intelligent decommitment, upon specific changes in the environment. It was argued that an appropriate selection, setting and preference ordering of decommitment rules contribute to robustness of the overall problem solution.

The task allocation and solution improvement using task delegation proposed in the work provides a powerful tool for problem solving while keeping the computational complexity within reasonable limits. However, this does not imply that the existing approach is without shortcomings, as limitations and admissibility constraints of the resource agents' optimization heuristics, which is crucial to the design and implementation of the multi-agent solver for a particular problem domain needs to be addressed first. An approach to plan representation based on social commitments they introduced, is also suitable for flexible re-planning and plan revision purposes in dynamic non-deterministic environments.

The extended abstract problem solving architecture addressed the issues of decommitment rules definition and their influence on the plan execution robustness and stability. Each of these rules provides a different impact on the agent's current state. For example, relaxation helps to maintain the commitment execution, delegation effectively unblocks the agent's resources and full decommitment releases the agent's resources by dropping the commitment. By implication, a combination of the decommitment rules is expected to emerge in a self-adaptation pattern that should lead to some sort of a real-time commitment execution optimization. The applicability of their multi-agent solver architecture is demonstrated on several real systems operating in the domains of vehicle routing problems, strategic mission planning, multi-robot frontiers exploration, and production planning.

2.8 Summary and Limitations of the Related Work

In this dissertation, a proposal to design an adaptive object-oriented scheduling system in a multi-agent environment for dynamic tasks scheduling was presented and discussed. The proposed framework is targeted at distributed heterogeneous computing system platforms. Although, Pinedo (1997) first introduced the architecture of object-oriented scheduling system, Pinedo, did not address the issue of controlling entities in a heterogeneous distributed environment, which is considered crucial for many domains. Pinedo, was also motivated by specific problem domain and design principle: that of the manufacturing industry and object-oriented scheduling method alone. The architecture Pinedo presented defines objects for every aspect of the scheduling system. As such, it promotes an extremely modular design which makes the system easily reconfigurable and extensible. This approach is not necessarily superior to other approaches as his design consideration can be used in conjunction with other scheduling approach to build a more robust and scalable system. In addition, Pinedo stopped short of integrating the proposed architecture into identifying common abstractions relevant to other related scheduling domains.

The intent of this chapter is to study distributed scheduling problem in a much broader perspective in such a way that the results can be made applicable in any distributed computing system area of interest (such as; cloud, grid, client-server, cluster, and peer-to-peer computing). Hindman *et al.* (2009) argued that in order to ease the development of new frameworks that transcends multiple domains and has the prospect of utilizing the same resource environment, it is critical to identify common abstraction and modularize their architectures. To achieve these goals within the premise of the proposed dissertation objectives, a proposal is made to develop a hybrid abstract scheduling architecture that comprises of two scheduling approaches, the object-

oriented method and multi-agent approach. The proposed abstract architecture provides guidelines, for isolation and efficient parallel job scheduling across frameworks, running on the same distributed computing environment, while letting each framework the freedom to implement its own programming model and fully control its scheduling decisions.

CHAPTER THREE

GENERAL-PURPOSE SCHEDULING FRAMEWORK

3.1 Synopsis of Chapter

In this chapter, a **conceptual design framework for scheduling systems** whose design methodologies and approaches are **object-oriented** throughout is introduced. General-purpose scheduling system architecture can be developed by leveraging some of the benefits that come with the application of object-oriented scheduling method and multi-agent scheduling approach. The proposed scheduling platform overview is discussed by presenting preliminary considerations for object-oriented design patterns, multi-agent technology and peer-to-peer network overlay platform. In this regard, an abstract scheduling architecture that is modular and adaptive in structure and that can also be sufficient for carrying out scheduling of parallel jobs within the context of distributed computing system environments is presented.

The chapter is organized as follows. In section 3.2, object-oriented design preliminary is presented. Section 3.3, focuses on the scheduling system framework and design pattern. This frame of modelling is geared towards the development of the modules necessary for dealing with the development of the proposed scheduling system. Section 3.4, presents and describes the proposed object-Oriented scheduling system architecture. Section 3.5, describes object-oriented patterns for interaction and collaboration of the entities described in section 3.3. Section 3.6, describes the architecture of the design patterns of multi-agent system that is combined with the model presented in section 3.3 to develop a complete scheduling system. Section 3.7, deals with multi-agent based communication patterns. Section 3.8, describes the dynamic object-based storage system design. Section 3.9, discusses techniques for achieving object components

reusability and portability for the general design model. And finally, section 10, summarizes the achievements made so far in this chapter.

3.2 Architecture of Object-Oriented Distributed Scheduling System

When developing solutions for large-scale scheduling problems, three approaches can be explored. The first option is to solve the problem by using a single scheduling system to get the optimal or near optimal solution, the second option is to carry out iterative scheduling simulation that produces approximate scheduling solution, and the third approach is to use distributed scheduling method. In this dissertation, the latter approach is adopted in combination with object-oriented techniques and multi-agent scheduling paradigm, to tackle issues related to design and architectural implementation of broad-spectrum scheduling system. The goal is to achieve these design goals by applying the methodologies of problem decomposition and solving techniques.

The concept of problem decomposition methodology and solving approaches, have been studied intensively by researchers for decades (Chiang *et al.*, 2007; Durfee, 1999). By using the problem decomposition methods, a problem is partitioned into sub-problems in which individual systems will require smaller amount of resources and information to solve the problems instead of demanding for huge amount of resource(s) at once. In this perspective, a distributed cooperative scheduling agent system could be designed where each system is responsible for a sub-problem and interacts with the other systems to construct a complete solution. However, in order to take advantage of the distributed system resources fully, the interaction between each system should be efficient and properly defined. Hence, the development of an effective communication infrastructure cannot be over emphasized.

In the proposed work, three design issues needs to be addressed, first, the system architecture, second, the communication infrastructures, and third, the coordination mechanism for distributed problem solving that handles the aspect of parallel jobs allocation by cooperative agents. Before setting up the communication channel for the multi-agent systems, first, the system architecture need to be constructed, followed by the communication patterns, and then, define the interactive protocols that would support this communication on both IPv4 and IPv6.

The architectural models presented in this section, gives first-hand information on some of the crucial structural issues that pertains to the proposed platform development process. Basically, the above model will serve as a primary tool for the design, evolution, implementation, management, migration and understanding of the proposed software system. The structural issues in this case include, gross organisation and global control structure; protocols for communication, synchronization and data access; assignment of functionality to design elements; physical distribution; composition of design elements, scaling and performance selection among design alternatives. The aim of this chapter therefore, is to develop the desired methodology, and where necessary, come up with the required tools that will support the architecture-based development process of the object-oriented multi-agent scheduling system. However, at this level, it is also ensured that all dependability requirements are fully met.

3.2.1 Object-Oriented Preliminaries

First, some of the properties of object-oriented design that makes it best option for the present research, are considered. Below are list of some of these essential properties (PALAI, 2011):

i. Abstraction: Abstraction consists of focusing on the essential, inherent aspects of an entity and ignoring its accidental properties. In system development, it focuses only on what an object is and does, before deciding how it should be implemented.

ii. Encapsulation: It can also be referred to as information hiding. It consists of separating the external aspects of an object, which are accessible to other objects, from the internal implementation details of the object, which are hidden from other objects. It is not unique to object oriented languages.

iii. Combining Data and Behaviour: The caller of an operation need not consider how many implementations of a given operation exist. Operator polymorphism shifts the burden of deciding what implementation to use from the calling code to the class hierarchy. As an example, let us talk about an object oriented program calling a draw procedure for drawing different figures, say, a polygon, circle, or text. The decision of which procedure to use is made by each object, based on its class.

iv. Sharing: Inheritance of both data structure and behaviour allow a common structure to be shared among several similar subclasses without redundancy. The sharing of code using inheritance is one of the main advantages of object oriented languages.

In following the principle and methodology of the object-oriented design concept, the initial system design is usually framed around two basic entities, namely, objects and methods. The object entities are more likely to be associated with sets of attributes that describe the object. Object in this case, refers to a data that can be quantified into discrete, distinguishable entities. For a typical scheduling scenario, the following components can be referred to as object entities or in order words, concepts that seems to represent an object; jobs on active queue, computer systems, computational nodes or distributed resource agents as the case maybe, scheduler agents, generated schedules, distributed system environment and many more. Sets of objects could at times possess

similar attributes, these objects thus, can be categorized into classes or object classes. Object Class or Class in this case, mean a description of objects with similar properties, common behaviour, common relationships to other objects and common semantics.

Methods are functions or transformations that can be applied to objects in a class. An example of this is the method *dispatch ()*: which is a function that is invoked by the scheduler agent object class to dispatch jobs into the resource agent environments for processing (Fig. 3.1). The attributes' component parts are usually those data values or set of static information held by the objects in a class. In the above example, quantity and capacity of a compute node are attributes. The is-a relationship and the has-a relationship, are two kinds of relationships that define relation between object types as mentioned in the literature and clearly illustrated in the work of Pinedo and Yen (1997).

In PALAI (2011), Object modelling is described as the abstraction of something for the purpose of understanding it before building it. Logically, it is far easier to manipulate an object model than the original entity. An object model captures the static structure of a system, relationships between the objects, and the attributes and operations that characterize each class of objects. This model is the most important one, as it will aid us through the actualization of the proposed research model. Some basic concepts covered in object modelling framework that have not been captured in this dissertation, but are deemed necessary include:

a. Object diagram: They provide a formal graphic notation for modelling objects, classes and their relationship to one another.

Fig. 3.1: Classes and Objects Diagram Model Structures

In Fig. 3.1, supposing a computer class is given, then, HP Pavilion can be referred to as an object of that class.

b. Class Diagram: It is a schema, pattern or template for describing many possible instances of data. This is often represented diagrammatically using rectangular shapes.

c. An Instance Diagram describes how a particular set of objects relate with each other. An instance diagram describes object instances. This is usually represented diagrammatically by rounded rectangular shapes.

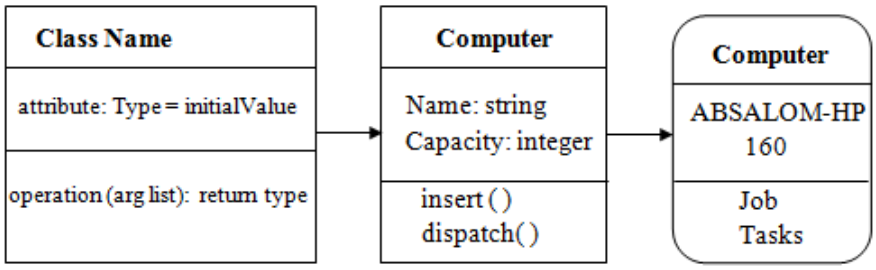
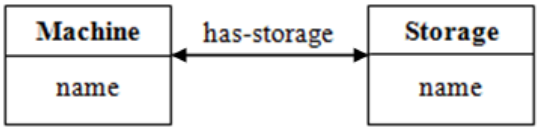


Fig. 3.2: Class with Attributes, Values and Method

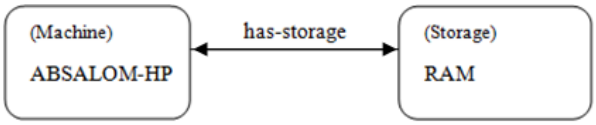
The first box in Fig. 3.2 illustrates classes with rectangles divided into compartments. The class name is contained in the first partition (centred, bolded, and capitalized), the attributes are listed in the second partition, while the operations are written into the third. Let consider an example whereby, a set of operations known as *insert* and *dispatch* is applied to the computer class with static information (attributes) name and capacity.

Two other significant components that complement the modelling structure are the Links and Associations. These two components are used for establishing relationships among objects and classes. Links are usually physical or conceptual connections between object instances, while association tends to describe a group of links with common structure and common semantics. They are inherently bidirectional, which

means that a binary association can be traversed. They are implemented as pointers. An instance of this is illustrated in Fig. 3.3.



a. Class Diagram



b. Object Instance Diagram

Fig. 3.3: Pointer Implementation of Links and Association

As relationships are established among objects and object instances, to trigger a change in state in such objects, an application of one or more operators is required. An operator can be regarded as the way in which a method is implemented in the software. Subsequently, the application of these operators on connected objects results in information or content flow from one object to another. An example of these contents that could be transmitted among related objects are schedule events. In other words, a message may also consist of attributes with their respective values or entire objects operational flows. Usually, this process often occurs in the cause of an event occurring at a particular stage of activity. Another kind of information that can be transmitted include agent's statuses (available to take on a job, busy, unavailable). However, this transmission approach requires the specification of protocols which would be looked into in the later part of this chapter.

3.2.2 Aggregation, Generalization and Inheritance

Aggregation is the “part-whole” relationship in which objects representing the components of something are associated with an object representing the entire assembly. An Aggregate relationship can be thought of as a "Consists Of" relationship, where the Whole consists of the Parts. For example, a Cluster can be seen as a whole entity and computational server or storage server as part of overall Cluster (Fig. 3.4).

It is equally important to clarify how defined classes are linked or related with one another through the use of the generalization and inheritance concept in OOP. Generalisation here, is seen as the relationship between a class and one or more refined versions of it. The referred to refined class is the '*superclass*' and each refined version is called '*subclass*'. As it applies to our case, *Distributed System* is a superclass of *Grid* and *Cluster.* Attributes and methods that are common to a group of subclasses are attached to the superclass and shared by each subclass. Generalisation is sometimes called a '*is-a*' relationship. Each instance of a subclass is an instance of the superclass. The discriminator (triangle symbol in Fig. 3.4), is an attribute of enumeration type that indicates which property of an object is being abstracted by a particular generalisation relationship.

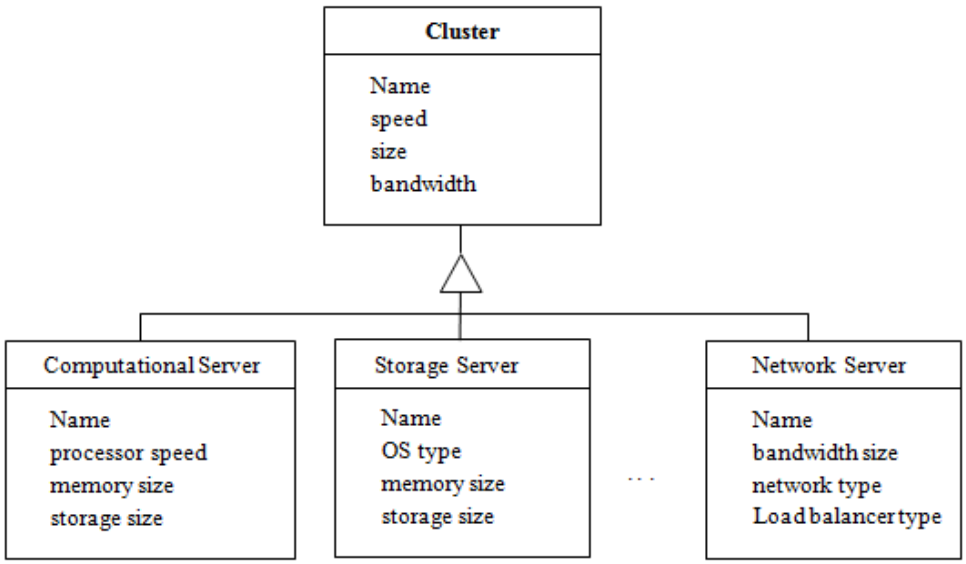


Fig. 3.4: Aggregation Relationship

3.2.3 Dynamic Modelling of System Behaviours

The notion that the study of temporal relationships is difficult to comprehend is quite true (PALAI, 2011). The easiest way to study and understand how a system works is by examining some of its static behaviours. However, the operation of a dynamic system in a dynamic environment is the primary concern in this case. Therefore, we examine the changes that manifest upon the system objects and their relationships overtime. In this section, emphases are placed more on those system aspects that are concerned with changes in the object and time. These concepts include events and states.

The attribute values and links held by an object are called the state of the object. Over time, the objects stimulate one another as a result of either internal or external influence, resulting in a series of changes to their respective states. An external trigger or individual stimulus from one object to another is referred to as an event. The response to an event depends on the state of the object receiving it, and can include a change of state or the sending of another event to the original sender or to a third object and so on.

An event is any operation that happens at any point in time, within the object life cycle. An example of this is a scenario where a task s, is to be scheduled or mapped to a resource r, for execution. An event has no duration. One event may logically precede or follow another, or the two events may be unrelated. An instance of this is where task s must be processed by resource r_1 before being executed by resource r_2. However, an event involving multiple objects can occur simultaneously, as the case may be for parallel processing of distributed tasks. In general, an event can be seen as a one-way transmission of information from one object to another. Although an object sending an event to another object may expect a reply, the reply is considered a separate event under the control of the second object, which may or may not choose to send it.

An event conveys information from one object to another. Some classes of events may be simply signals that something has occurred, while other classes of events convey data values. The data values conveyed by events are their attributes, like data values held by their objects. Attributes are shown in parentheses after the event class name.

event class name (attributes)

server execute (OS, processor, memory, storage)

A state is an abstraction of the attribute values and links of an object. The set of values are grouped together into a state according to properties that affect the gross behaviour of the object. For example, let us consider the state of an agent system (an object). It can either be in available, busy or unavailable state depending on it processing power. A state specifies the response of the object to input events. The response to an event may vary quantitatively depending on the exact value of its attributes. The response is the same for values within the same state but different for values in different states. The response of an object to an event may include an action or a change of state by the

object. Example, a process is in ready state when it can accept a task, but it changes from ready to busy state during the execution of the accepted tasks.

A state corresponds to the interval between two events received by an object. Events represent points in time, while, states represent intervals of time. For example, in the task process example, after the task is assigned and before execution begins, the process is still in a ready state. The state of an object depends on the past sequence of events it has received, but in most cases, past events are eventually hidden by subsequent events. For example, events that have happened before the execution of previous processes have no effect on future behaviours of incoming processes. Next, a concrete example of state behaviour for a Grid job execution is given.

In the course of job scheduling process, a job passes through several states (Venugopal *et al.*, 2008), as outlined in Fig. 3.5. A job is an input to the Scheduler which allocates it to a set of resources based on its resource requirements profile. The first state of any user job is the "ready" state, after which the status is changed to "scheduled". During the "stage in" state, which is at the point when the job is to be admitted into the remote scheduling process, input profile requirements for the job is staged into the remote resource. When this process is completed successfully and a handle is obtained, then a job is considered to be finally "submitted". The job may be queued while waiting for an available processor and its state changes to "pending". When the job starts its execution, it is considered "active". After the job has finished executing, it enters the "stage out" stage where its output profile is transferred back to the resource provider. If all its output profile is received and is as expected by the task requirements, then the job is considered as "done". If one of the state transitions fails on the remote side or the job has completed on the remote side but has not produced the expected result profile, then it is considered "failed" and is reassigned again for rescheduling. Also, if the job was

interrupted and could not complete the execution process or it is terminated prematurely, then it is considered “aborted” and is re-set for scheduling.

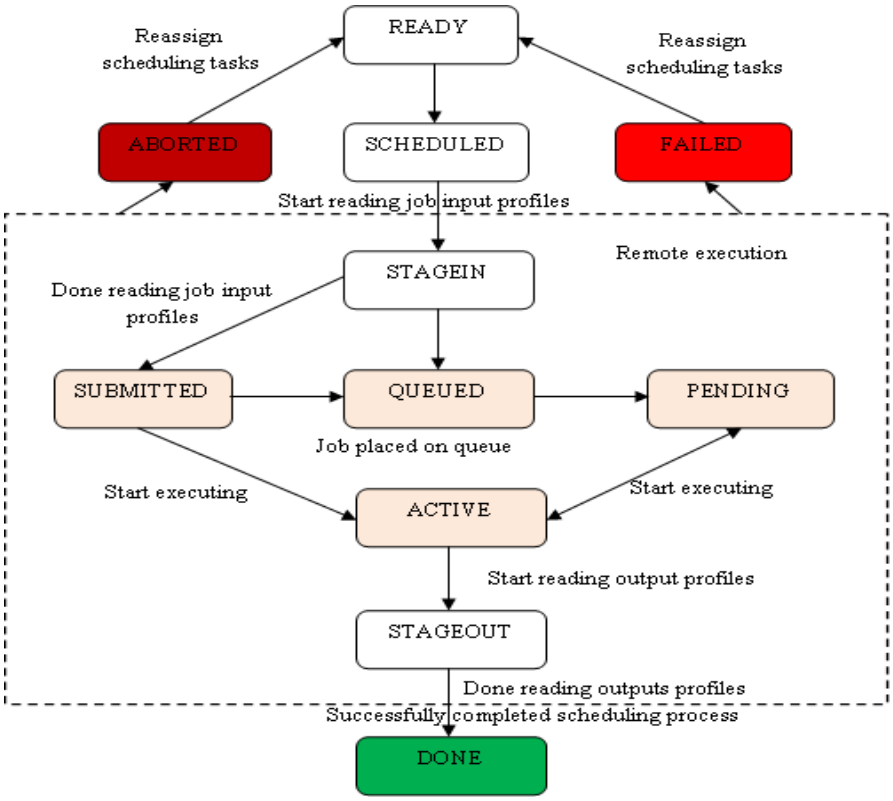


Fig. 3.5: Grid Job Execution State

Previously, an illustration of the abstraction of the modelling tools described Fig. 3.5, with their respective applications to designing more complicated relationships between objects was presented. Much emphasis have been given to the various activities of the modelling components' key players, such as objects, attributes, methods or operations and contents conveyed as a result of interactions among objects (here referred to as messages). Figure 3.5 somewhat depicts these points of multiple relationships and interactions that could exist between objects and object instances. In the model, introduction of some of the more specialized object entities that often times would assume different roles is made. A typical scheduling system however, would comprise of several of these specialised object entities in the form of user jobs, computer systems

or nodes as resources, etc. as well as one or more operations. The parameterised operation usually carries some contents or messages transmitted from one agent object to the other that causes changes in state of particular objects. This change in state of object subsequently would lead to coordinated execution of tasks being assigned to the system. Messages are usually expressed in the form of parameterised expressions with attributes serving as the contents. Just like classes, attributes likewise can be listed in a separate compartment. However, attributes when expressed, must be accompanied with assigned values. Fig. 3.6 depicts this illustration.

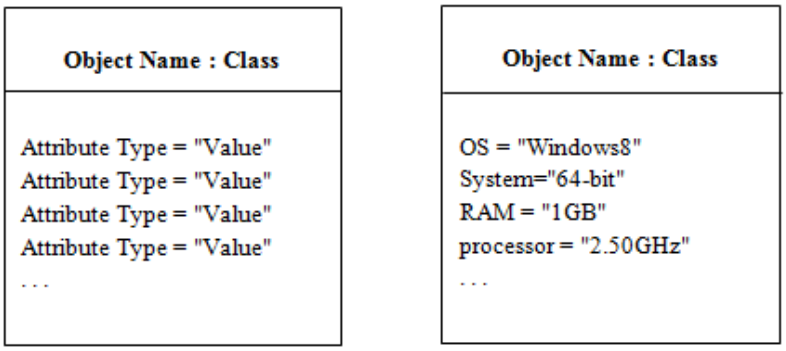


Fig. 3.6: Objects Attributes Representation

Interaction among object classes in terms of exchange of messages overtime could be expressed in the following way:

Object.methodN(messageN) or Object.operationN(messageN)

server.execute (objectname, variable, ...)

For emphasis sake, by message here, it means those solid lines or arrow lines that depict communication links between objects. These arrow lines could be seen in most of the model diagrams presented in this chapter.

3.3 Scheduling System Frameworks and Design Patterns

The global intent of this dissertation is centred on the marriage of two design principles, frameworks and design patterns. In this section, best practice solution to the reoccurring scheduling problems common to the existing scheduling frameworks is proposed and discussed.

3.3.1 Scheduling System Frameworks

Frameworks are an object-oriented reuse technique. They share many characteristics with reuse techniques in general (Krueger, 1992), and object-oriented reuse techniques in particular. Although they have been used successfully for some time, and are an important part of the culture of long-time object-oriented developers, they are not well understood outside the object-oriented community and are often misused (Johnson, 1997). Frameworks generally comprises of the combination of components and patterns. Although they can be thought of as a more concrete form of a pattern, frameworks are more like techniques that reuse both design and code.

One of the strengths of frameworks is that they are represented by traditional object-oriented programming languages. However, this is also a weakness of frameworks, and it is one that the other design-oriented reuse techniques do not share. The implication of channelling the design of frameworks toward a particular object-oriented language, restricts frameworks to systems using that language. In general, different object-oriented programming languages do not work well together, so it is not cost-effective to build an application in one language with a framework written in another.

An alternative solution is to incorporate patterns into frameworks, so as to properly document and interpret collaborations in frameworks. Another way to strengthen the loop-holes in frameworks, is to describe collaborations between objects in terms of

component structures. Clear descriptions of all classes, interfaces, and methods representing the basic components of the proposed general-purpose scheduling framework are discussed in section 3.4.3 and depicted in Fig. 3.9.

3.3.2 Scheduling System Design Patterns

Design patterns have been used to identify, name and abstract common themes in object-oriented design. They capture the intent behind a design by identifying objects, their collaborations, and the distribution of responsibilities (section 3.4.3). And when augmented with the activities of autonomous intelligent agents, can provide adaptive scheduling solutions. Design patterns play many roles in the object-oriented and multi-agent development process; they provide a common vocabulary for design, they reduce system complexity by naming and defining abstractions, they constitute a base of experience for building reusable software, and they act as building blocks from which more complex designs can be built. Design patterns can be considered reusable micro-architectures that contribute to overall system architecture. In this section, a description of how to express and apply design patterns to the design of general-purpose scheduling framework is discussed.

Just as design patterns have many applications in the object-oriented development process (Gamma *et al.,* 1993), they equally have tremendous uses in the planning and design of robust scheduling systems architectural frameworks. Some of these uses can be expressed as follows:

a. Design patterns have shown beyond reasonable doubt how to provide a common vocabulary for framework designers and software developers to communicate, document, and explore different design alternatives. Especially in identifying the

best scheduling alternatives and components with respect to the targeted applications and systems environment at the very beginning of system design.

b. Design patterns constitute a reusable base of experience for building reusable scheduling software. They distil and provide a means to reuse the design knowledge gained by experienced application domain practitioners. Design patterns act as building blocks for constructing more complex inter-domain or inter-disciplinary scheduling designs; they can be considered micro-architectures that contribute to overall scheduling system reference architecture.

c. Design patterns provide a target for the reorganisation or refactoring of class hierarchies (Opdyke, 1990). Moreover, by using design patterns early in the modelling of the scheduling system lifecycle, one can avert refactoring at later stages of the system design.

d. Design patterns provide much of the design work upfront. A well-written scheduling pattern for instance, discusses in detail the problem and issues that need to be solved with regards to the scheduling components involved and show how the problem is solved, with a good discussion of the pros, cons, and other issues to be aware of. By reading a pattern, many challenging and potentially hidden issues in distributed system environment can be discussed and considered upfront.

e. Design patterns are one approach that can be used to mitigate some of the primary limitations of traditional object-oriented design (Monroe *et al.*, 1996), two of these limitations referred to here are; the difficulty involved in specifying how groups of objects (such as jobs, schedulers or mediators and machines or resource) interact and in specifying and packaging related collections of objects for reuse.

The integration of design patterns together with frameworks into our work is to capture the static and dynamic aspects of successful solutions to problems that

commonly arise when building a general-purpose scheduling system. The integration of design pattern principles in this case, can help enhance the quality of the proposed scheduling framework, by addressing fundamental challenges in large-scale distributed system scheduling environments. These challenges include, communication of architectural knowledge among developers of different scheduling models, accommodating new design paradigms or architectural styles associated with application-specific scheduling domain, and avoiding development traps and pitfalls that are usually learned only by experience.

3.3.3 Scheduler Pattern

The Scheduler Pattern (Fig. 3.7), take the action of decoupling user application from the remote resource domains. An object scheduler is an object which knows the location of other objects (either job objects, scheduler objects or resource objects). The scheduler can have the knowledge a priori (at compile-time) or can gather the information dynamically as objects register themselves or a combination of both. The primary advantage of the Scheduler Pattern is that, it is possible to construct a Proxy Pattern when the location of the resource is not known when the system is compiled. The proxy encapsulates the information necessary to contact the real object and get up-to-date object information.

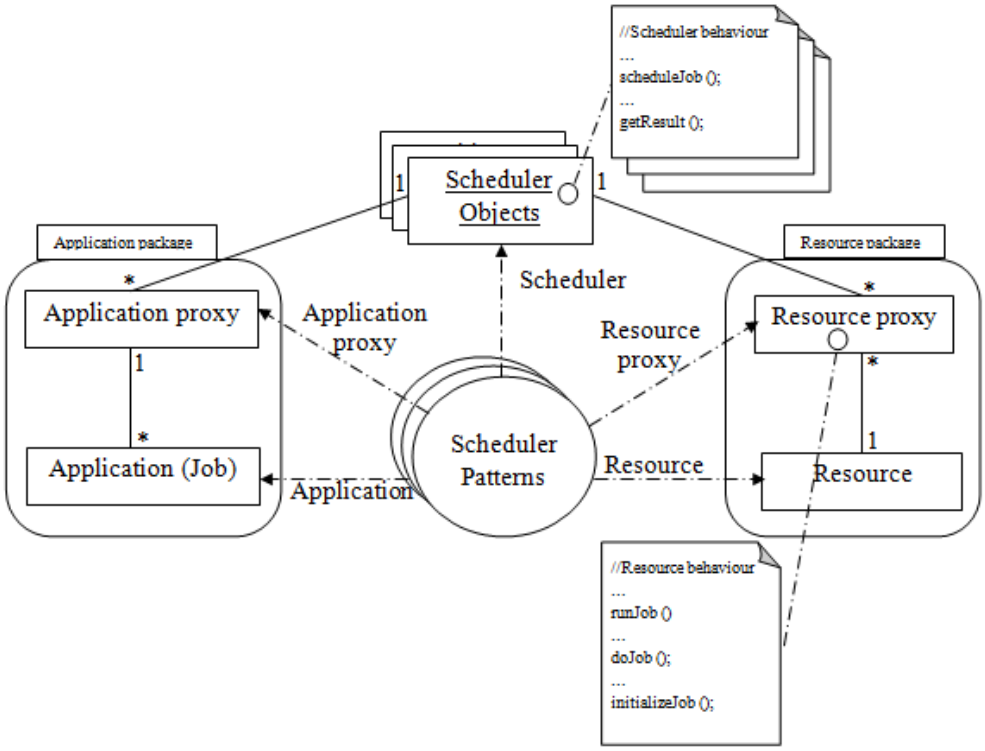


Fig. 3.7: Scheduler Pattern

3.4 Proposed Object-Oriented Scheduling System Architecture

This section begins with the description of the proposed object-oriented scheduling system, by discussing the overall design philosophy in terms of object-oriented design. However, the aim of the object-oriented design (OOD) is to design the system in the form of objects, that is, instantiations of the classes and subclasses extracted during object-oriented analysis phase. First, a general structural workflow diagram representing the object-oriented scheduling system is presented in Fig. 3.8. In this diagram, the various components that collaborate with one another to form a complete system are depicted. The presentation and conversion of the scheduling elements into object-based entities and instances are likewise presented here in the form of object-orientations and objects instantiations model; this is shown in Fig. 3.9. The top down design model is followed in describing the general design overview for the proposed object-oriented

scheduling system architecture. The remaining part of the section describes and discusses the design workflow of the unified scheduling process.

3.4.1 Object-Oriented Design Philosophy

The proposed scheduler aims to provide guidelines that facilitate and standardize the design and the development of a scheduling system whose design concepts are based on the object-oriented design and programming approach. One major setback with most of the heuristic-based scheduling system is lack of code and software component reusability, and these invariably restrict portability, adaptability and scalability in those schedulers.

There are numerous advantages in following the object-oriented design approach in developing robust scheduling system that is adaptive and scalable. One of these advantages is that the design is usually modular, which makes maintenance, modification and configuration of the system relatively easy to implement and deploy. Another merit of this approach is that, implementation resources such as coding and documentation can easily be reused when the need arises. What this simply means is that, two or more scheduling systems which might differ substantially, can still share a significant amount of codes and other relevant implementation resources. The object-oriented design approach tends to hasten up design process, and separates the design process from the implementation process by making the developer think only in terms of object behaviour rather than low-level details.

3.4.2. System Architecture Overview

The object-oriented scheduling model follows a very popular software engineering design concept, the modular based system development architectural framework, in which the scheduling system is composed of different object-oriented independent

entities which collectively implement the desired scheduling functionality (Fig. 3.8). The system consists of the parallel job selector object, decentralized scheduler objects, object-based store (which stores up-to-date objects behaviour and states), resource information object (keeps up-to-date information regarding the current resource status, stores information about jobs currently being executed and those awaiting execution based on previously generated planned schedules) and distributed resource(s), each with its respective local allocation policies.

Intelligence among objects is initiated by the ability of individual objects to communicate with others and share processing responsibilities together. The implementation of various methods defined on specific objects, causes information to be transmitted from one object to the other, and this often triggers communication among object entities. Communication somewhat represents information or messages that are transmitted from one object (for example a cluster) via a method to another object (for example job schedule planned for next execution). A communication may consist of simple static information (job objects attributes or resource objects specifications) or dynamic information (such as specific behaviours and states associated with objects in the course of an event trigger). Communications between objects usually occur at the time those events have occurred (which is caused by the application of operation to objects).

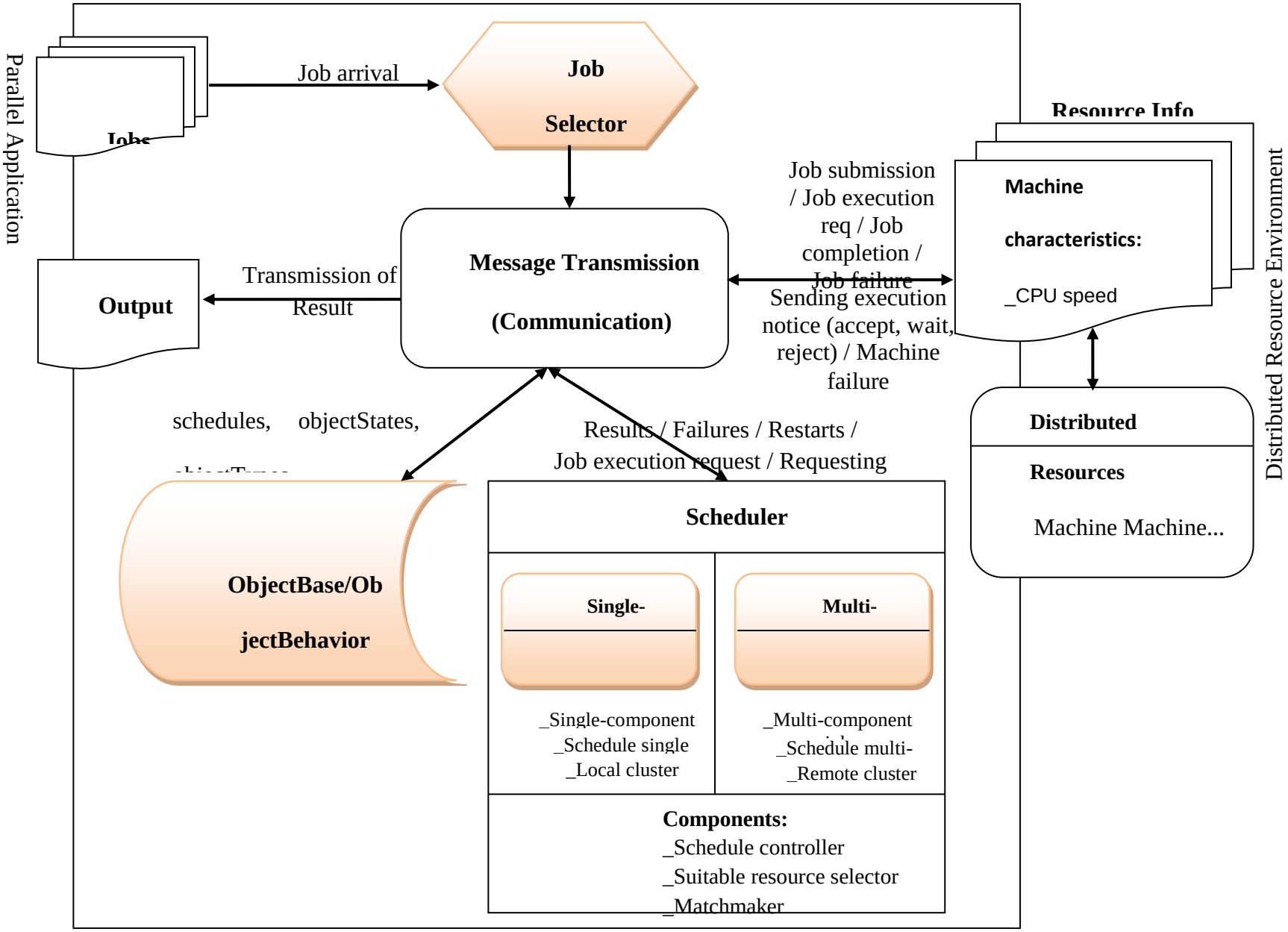


Fig. 3.8: Object-Oriented Scheduling System Workflow Overview

3.4.3 Object-oriented Scheduling System Design Pattern

In this perspective, a design pattern is viewed as a solution to a general design problem in the form of set of interacting classes or modules that have to be customized to create a specific design framework. The Object-oriented scheduling system has a modular architecture grouped into three main layers. The first layer is the parallel job layer that comprises of components which are responsible for job profiling and job loading into the system. It is also at this level that decision is made about which scheduler is selected based on suitability for a particular job resource requirement (that is, whether a job would require multi-cluster or single cluster resources based on job structure).

The second layer is the scheduling layer, which communicates with the external distributed resource environment (namely, grid, cloud, etcetera.). This layer is majorly responsible for general job scheduling, data and scheduling process management. It can be referred to as the scheduling system kernel. Modules from this layer also handle the movement of parallel jobs from the schedulers to the distributed resource gateway system, and collects feedback information (of the processed data) and computes result of the user jobs.

The third layer consists of the physical resources comprising of all computational nodes, storage disks and network resources. The distributed resource gateway, distributed resource information service and resource management system provide information regarding published resource characteristics (such as OS type, CPU speed, available RAM, the associated input/output communication bandwidth and disk space or the availability of certain software in real time) and allows for the execution of user's job by remote resources on a wide variety of distributed applications.

The *design pattern* depicted in the model presented in Fig. 3.9 is divided into three categories; creational design patterns, structural design patterns, and behavioural design patterns. The *creational design pattern* solves design problems by creating objects; the abstract object-oriented scheduling system is an example (job objects, schedule objects and resource objects have been created for the new system). The *structural design pattern* solves design problems by identifying a simple way to realize relationships between entities. An example includes established relationships among different job components and resource components. This is also reflected in the structural model in Fig. 3.9. Lastly, the *behavioural design pattern* solves design problem by identifying common communication patterns between objects. An example of this type of design pattern is the objectBase and objectBehaviour patterns shown in the following model.

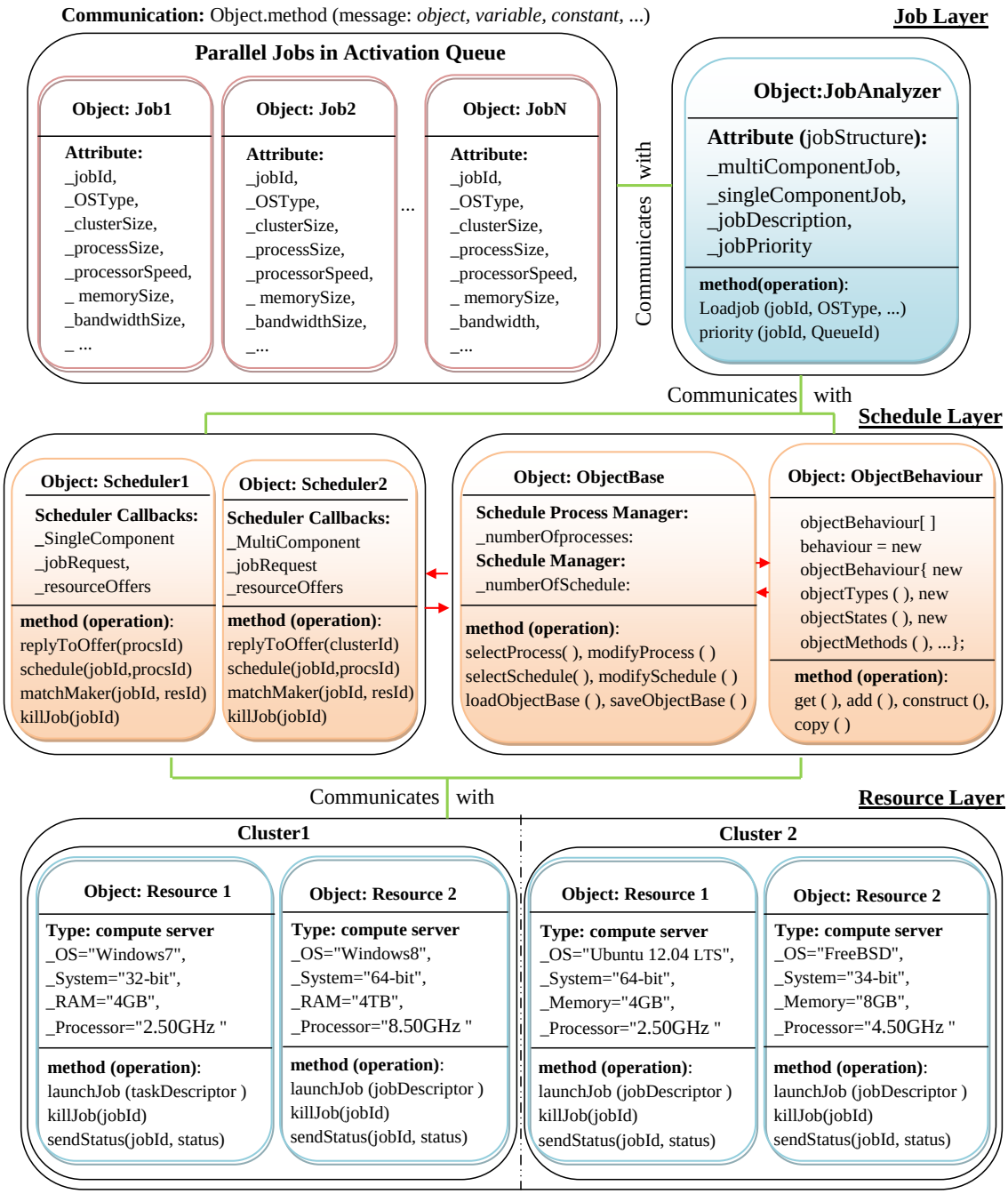


Fig. 3.9: Object-Oriented Scheduling System Design Pattern

The arrival and processing of parallel jobs in the activation queue follows a First in First out (FIFO) queuing policy. Processes are dispatched according to their arrival time on the ready queue. The structure of the job is very significant to the processing of each job by the job analyser module, followed by the schedulers' event. However, in line with this condition, a job can specify its resource requirements in three categories; required

resources, interactive resources and optional resources (Bhama and Selvi, 2011; Czajkowski *et al.*, 1999).

In our model, instances of resources coming from unit cluster and ones coming from multiple clusters originating from different or the same resource providers (remote cluster) are considered. For those jobs requesting the utilization of multiple clusters, the concept of resource co-allocation strategy in distributed computing environment (cloud, grid, etcetera.) is applied. Resource co-allocation is the technique where there is a simultaneous or coordinated access by a single application to resources of possibly multiple types in multiple locations managed by different resource managers (Bhama and Selvi, 2011). Any job that requires the use of multiple resources managed by different providers would have to specify the set of tasks and number of processes that have to run on the different clusters.

The scheduler module shown in Fig. 3.10 handles the scheduling of profiled and prioritized jobs. This module is further divided into two sub-components, the *single component scheduler* that deals with the single component jobs requesting for multiple processors within a unit cluster and the *multi-component scheduler* that handles multiple component jobs which requires multiple processors from different clusters coming from different resource providers' sites (this can be likened to the multi-site co-allocation strategy). The two schedulers are designed around two basic entities, namely, *objects* and *methods*. The work of each scheduler is concentrated around searching for suitable resource for diverse users' applications. A compromise must be reached between the application resource needs and the available resource, after which the scheduler matches the job with the candidate resources and allocation made to execute the processes.

However, scheduling multi-component jobs is more complex than scheduling single-component jobs. Multi-component scheduler would basically have to put into consideration a factor due to jobs time dependency. Another added burden to the multi-component scheduler is that, it has to analyse more mapping options as some applications will have more flexibility on how to map jobs to resources. Network overhead is another issue that the multi-component scheduler would have to handle especially for parallel jobs with inter-process communication. These issues can be addressed by incorporating into the design, mortalities such as the use of network information, proximity of data location to resources, network-aware scheduling, rescheduling and negotiation support.

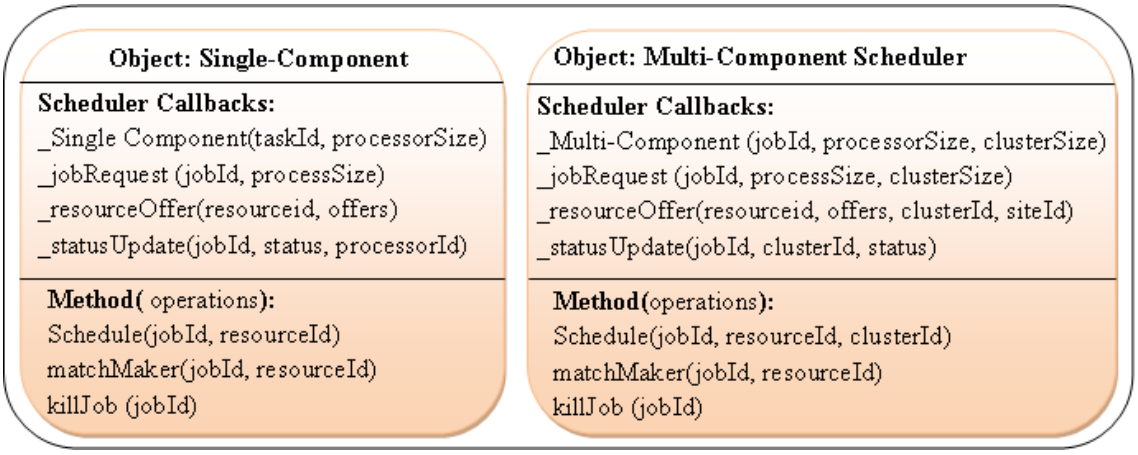


Fig. 3.10: Object-Oriented Scheduler Showing Two Types of Scheduler Components

The design strategy adopted in this work has the following strengths:

a. The design pattern provides high-level documentation of the design, simply because it specifies design abstractions.
b. Implementation of the model is very much feasible and as such, little effort is required to code or document those parts of the program that implement the

proposed design framework. However, testing is very much necessary to ascertain its performance output.

c. If a maintenance programmer is familiar with the design patterns, it will be easier to comprehend a program that incorporates design pattern, even if he or she has never seen that specific program before.

3.4.4 Object-Method Interactions

In section 3.2.1, it was mentioned that object-oriented systems are usually designed around two basic entities, namely, *objects* and *methods*, and by object, it refer to various types of entities or concepts such as grid resources (like computational nodes, storage servers, network server, etc.) and parallel jobs. However, objects usually possess some information that could be static (known as object attributes) or dynamic (referred to as object states). Objects can possess a number of these attributes and states at the same time. The application of a method to an object may cause information to be transmitted from one object to another (right in the heart of the scheduling system is the communication network module that links all the system components together, Fig. 3.8).

In this section, explanation of some of the most important methods as applied to the key system components shown in Fig. 3.9, which are required in order to change processing sequences is made. Starting with the object-based module, the *loadObjectBas*e method retrieves copies of stored schedule object instances previously generated by the schedule event. The *saveObjectBase* method saves a copy of the same schedule processes generated by the schedulers at every point in time. The *get* method fetches from the object behaviours' module any one of the object behaviours required by any of the colleague object processes for coordinated processing. The *add* method adds to the

already existing lists of object behaviour a new object behaviour type. The *construct* methods create new object behaviour and add such to a list of existing ones. The *copy* method creates a replica copy of any of the existing object behaviour.

The object-based module also consists of two other major components that control the interaction of the scheduler with the non-scheduling object components of the general system. These components are the *process manager* object, this object manages a number of different processes that holds information regarding distributed system resource objects, sets of user jobs object entities, processing constraints or objective functions. The *schedule manager* object is an object class that manages a number of other associated schedule objects. It stores detail information regarding a particular number of associated schedules, whether it is a complete schedule or partial schedule. These numbers of associated schedules can in reality be either, resource schedule, time schedule, etc.

The following three methods associated with the scheduler module are concerned with the scheduling of tasks and operations. The *schedule* method triggers an event that generates scheduling plan for job execution and other scheduling events that are stored in the object base for subsequent retrieval and processing. The *matchMaker* method matches suitable discovered resources with ready jobs by generating logical mapping and associating it with scheduled jobs. The *killJob* method destroys any schedule process that could not meet up with the standard allocation policy, while those job schedules that meet the allocation policy are left unharmed and are assigned resources normally.

The method that finalizes the overall processes is the *executor* method which is concerned with the execution of successfully scheduled jobs. The implementation of

this active operation depends on the scheduler being able to discover suitable resource for every user jobs submitted to the system for execution. This method is implemented on every active node. Active node here, mean a physical running machine that a code segment can be executed on. The main idea behind the proposed scheduling model is to implement a broad-spectrum scheduling platform that is able to efficiently allocate distributed computing resource to parallel application jobs.

3.4.5 Abstraction of Scheduling System Design Pattern

Generalizing the overall solution to the intended complete system framework leads to the scheduling system design pattern shown in Fig.3.11. In this figure, a general solution to the problem of permitting communication between two objects with incompatibility interfaces is depicted. This illustration provides a way for an object to permit access to its internal implementation in such a way that external objects are not coupled to the structure of that internal implementation. One good side to this communication approach is that, it provides all the advantages of information hiding without having to actually hide implementation details.

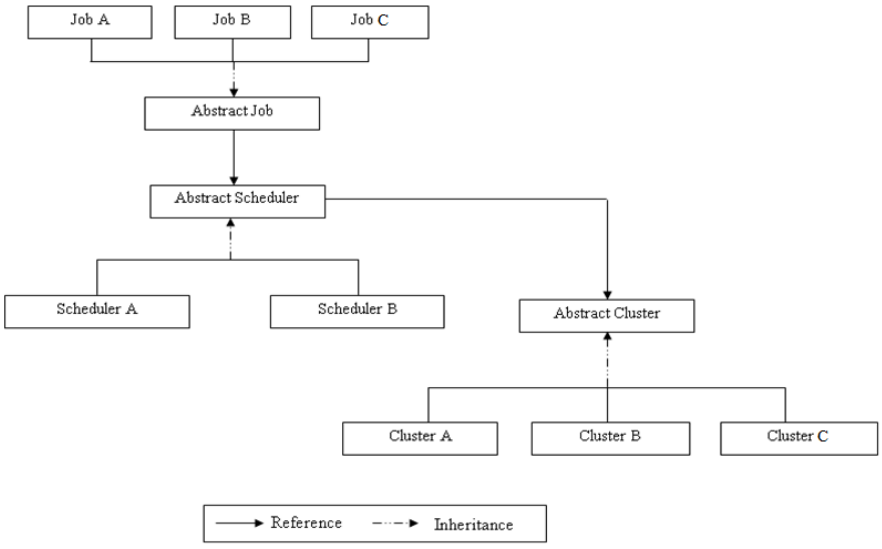


Fig. 3.11: The Scheduling System Abstract Design Pattern

Creating objects in terms of classes and methods, abstraction can be defined on these two mechanisms by creating abstract (virtual) classes together with their abstract methods. Abstract classes in this case, imply those classes that cannot be instantiated, although they can be used as base classes. An abstract class usually contains at least one abstract method, that is, a method with an interface but without an implementation.

3.4.6 Job Object Scheduling Process

The most interesting aspect of the scheduling process model is the part that handles the scheduling and execution of parallel jobs, starting from the point of submission to the point when the job is successfully executed. These jobs are programming code specifications, which contain specific set of instructions, which are required to execute the jobs. An example of job instruction, contains detail descriptions and specifications of the type of resources (such as; CPU speed, number of processors, estimated usage time, memory, associated bandwidth or other specialised resource), required to execute a job. A general diagram describing the job scheduling process flow-path is presented in Fig. 3.12. The modular structure of the model presented here follows the functional cohesion design technique. The functional cohesion modularization encourages module reusability. Also, in Section 3.4.7 and 3.4.8, a detailed description of both resources and application models used in this dissertation is presented.

The diagram below is made up of both shaded and non-shaded blocks. The shaded blocks define the modules which are to be implemented as part of our work. The non-shaded blocks are the external modules. These external modules provide a plug-in mechanism by integrating available external distributed resource components that are localized to the service providers.

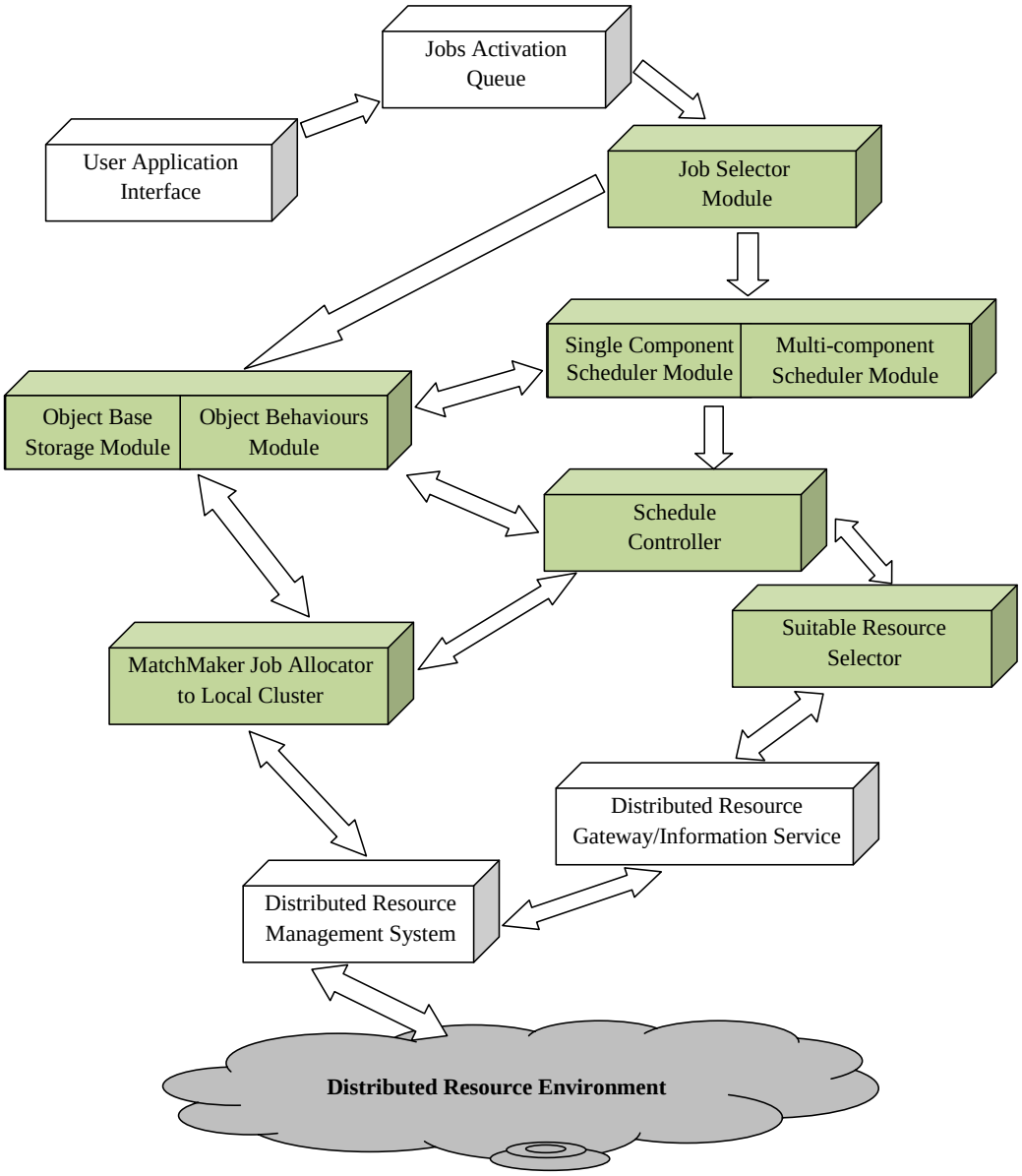


Fig. 3.12: The Distributed Jobs Processing in the Object-Oriented Scheduler

A. Job Selector Module

The job analyser module performs the function of analyzing the structure of the users' job and assigning priority based on which job comes first or enters the queue first. The task analyser further classifies the jobs into single component jobs, depending on whether a job requires just single cluster resources to execute its processes and multi-component jobs if the jobs would require multiple cluster resources to execute its processes. The cluster resources can come from any resource provider domain (a

provider contains clusters with a set of processors) or sites (site is the physical location of the provider). Job (or request), implies parallel applications that would ordinarily require sets of processors for execution. The Job analyser component, apart from analyzing the structure of a job, would also be engaged in searching for the most suitable scheduler using a desirable search policy to delegate any one of the selected jobs to the scheduler.

In this dissertation, a job is described by C integers, at least one of which $C \neq 0$. Two cases of job structure are considered, one for multi-clusters and one for single clusters:

a. For an unordered job resource request, with $C > 1$, a job is required to specify the resource configurations (number of processors, CPU speeds, memory sizes, and associated bandwidths), it needs in the separate clusters, allowing the scheduler to choose the clusters for the components.
b. For a job resource request, with $C = 1$, a job specifies only the total number of processors, CPU speeds, memory sizes, and associated bandwidths it needs, which is obtained as the sum of the C integers.

B. Component-Based Scheduler Module

The component-based scheduler is further divided into two, single component scheduler and multiple component scheduler. The single component scheduler handles single component jobs that require single cluster resources. The single component jobs are not usually most compute intensive application and as such can be executed by single cluster resource. The multi-component scheduler deals with the multi-component jobs dispatched by the job analyser module to the two schedulers. The multi-component jobs require multiple cluster resource(s) for their execution. These are often very compute intensive parallel applications requiring huge amount of processing power which might

not be satisfied by just using single cluster machines alone. However, a mapping policy that will select more of the suitable cluster resource for the parallel jobs is required here.

In the proposed architecture, a multi-cluster system consists of C clusters of possibly different numbers of processors, memory sizes, CPU speeds, and associated network bandwidths, in which all have different service rates. When $C = 1$, the system is termed to be single cluster. Otherwise, when $C > 1$, the system is said to be multi-cluster.

C. Schedule Controller

The schedule controller controls every scheduling activity that is ongoing within the internal organisation of the scheduling system. It also handles knowledge and information transfer regarding discovered resources and delegation of available (suitable) resource to the schedulers and other scheduling modules, namely, the suitable resource selector and match maker. The schedule controller controls every scheduling related activity that occurs during job scheduling processing. It also updates the states of the currently running process, job and resource status into the object base and object behaviour module concurrently.

D. Suitable Resource Selector Module

This module feeds the schedulers with information about the most suitable resources published by the resource providers through the distributed resource gateway and resource information service external module. The work of the distributed resource gateway/information service is to send job descriptions to the distributed resource management system, offer information regarding available resources together with their individual local scheduling policies to the schedulers, and provide lists of available neighbouring or remote clusters. The distributed resource management system is an

external Meta-scheduling system, responsible for locally scheduling and executing jobs using the distributed system computational resources.

E. Matchmaker Module

The Matchmaker module receives jobs information, with their generated schedules transferred from the schedule controller module, evaluates the adopted scheduling policy with the knowledge of local resource capabilities, and decides whether the job could be executed locally. The matchmaker contacts the distributed resource management system for the implementation of this action. If job allocation to the local resources is possible, then the matchmaker allocates the job to the distributed resource management system for execution. However, if a job cannot be executed by the local resources of a chosen cluster, based on the information received from the resource management system, the matchmaker sends request for an appropriate remote cluster to the resource selector module via the schedule controller. A cached list of remote cluster is provided by the distributed system resource information service which is exposed to the resource selector module.

F. Object Store and Behaviour Module

Another significant component part is the object behaviour module; this module can be likened to an array that keeps updated collections of object types, object states, object methods, object behaviours, and other related functionalities possessed by an object instance. Objects usually carry along dynamic information, usually referred to as the state or event of the object. An object may be in any one of the states. A typical instance of this is that, a machine can be in any of these states, namely, active, off, or failure. This module also makes it possible to dynamically create and add objects among the list

of already existing ones. The method applied to this module are the, *add, construct, copy* and *get* operation. An example of this is shown in the code listing segment below:

```
objectBehaviour[]behaviour = new objectBehaviour{ new objectTypes(), new objectStates(), new objectMethods(), ...};
```

To access the array object, the object need to be casted back to the objectBehaviour class, this is written in the form *objectBehaviour[key]behaviour.get()*, where the key can take an integer value or strings. Figure 3.13 shows our representation of hierarchy of relationships between the Object behaviour class and its components.

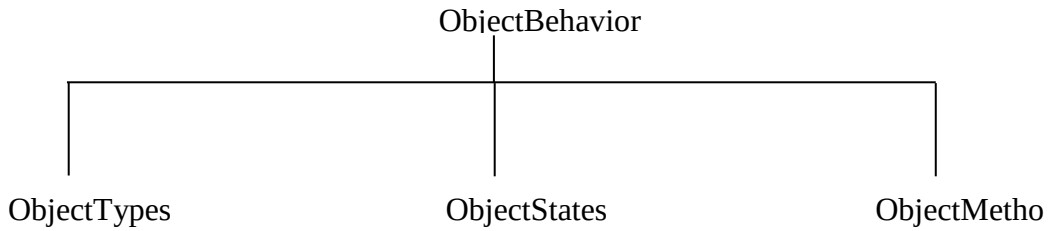


Fig. 3.13: Object Behaviour Relationship Hierarchy

However, the object behaviour class could still be defined and referenced in terms of parallel array definition. For instance, separate arrays can be defined for each of the object behaviour components (objectTypes, objectStates, and objectMethods), or even create an array list. Either way will still give a good access to the whole array. An example of an array code listing is given below:

```
ArrayList objectBehaviour = new ArrayList();
for(int i=0; i<12; i++){
int[]  objectTypes = new int[12];
objectBehaviour.add(objectTypes);
}
```

After this, objectBehaviour.get(0) is one of the array, objectBehaviour.get(1) is another array and the process continues up to objectBehaviour.get(11).

In a distributed environment, such as cloud or grid, system information often does not last for long, mainly due to the fact that users' jobs requirements and participation are dynamic and may change frequently in line with different scheduling policies (requirements).

System failure might be a leading cause of change in system information. The proposed scheduling platform provides support in which the state information of all objects is preserved, in case of any eventualities (such as machine failure) which might occur. The object behaviour module handles this by providing useful information regarding the states of all scheduling object components and instantiations in the system environment. The local search algorithm has been used to often rectify incorrect schedules resulting from machine failure event (Klusacek *et al.,* 2007). The object behaviour module, hopefully, can provide useful information that can precisely identify which jobs were affected so that they can be rescheduled onto other available neighbouring machines.

G. The Object Base Storage Module

The object base storage module hierarchically stores and manipulates the schedule objects. The module is very significant to the scheduling system in general, as it stores definitions of all major object types, namely, machine object and job object. In essence, the object base storage could be seen as the object library. An operation implemented in each of the instantiated object could also be stored in this same module.

The essence of this research is to describe a conceptual design for an object-oriented scheduling system that completely follows an object-oriented design principles and approaches. The object-oriented design methodologies adopted extends to all the modules that have been discussed in this chapter and the previous ones. In other words, object-oriented system design is considered while regarding the representation of the

scheduling problem, the mechanism for solving the problem (which is based on the representation), and the structure of the system supporting the mechanism as one entity.

H. Job Scheduling Flow Pattern

The job scheduling process needs to pass through different phases before reaching the execution stage. Each of these transition stages are envisaged and specified here. They are described in Fig. 3.14. Figures 3.14a and 3.14b are the products of the different job structures described earlier on. Two types of parallel jobs are assumed, single component, which is processed by single component scheduler (Fig. 3.14a) and the multiple component jobs, which is handled by the multi-component scheduler (Fig. 3.14b).

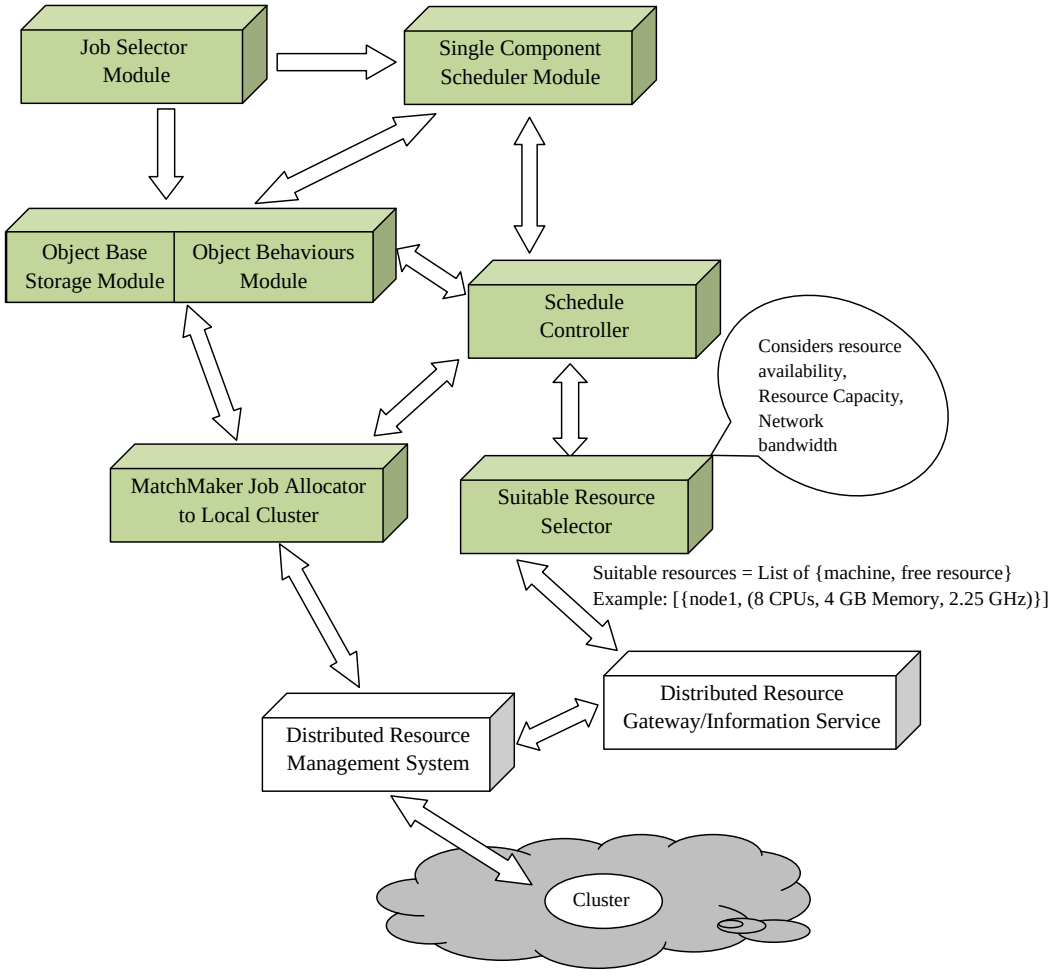


Fig. 3.14a: Single Component Job Scheduling

The first stage involved in the job processing and scheduling activity is the job loading and analyzing procession. Jobs have to be made available in the activation queue, after which they are picked and analysed by the Job analyser module. The detail description is given below:

a. Job is submitted on queue and prioritized based on first in first out basis.
b. Job is loaded into the job selector module by the job loader (which is not described above).
c. The job selector profiles the job to check whether the job is either;
    i. single component job or
    ii. multi-cluster job
d. Next, the job selector registers the job resource requirements information and other associated abstract queries from the user job into the object-storage and searches for an appropriate scheduler and delegates the job to it.
e. The selected scheduler is either:
    i. Single component scheduler or
    ii. Multi-component scheduler.
f. The scheduler schedules the job by:
    i. Querying the schedule controller for information regarding available resources.
    ii. The schedule controller in turn queries the resource selector module.
    iii. The resource selector in turn, queries the distributed resource information service via the distributed resource gateway. Both resource gateway and resource information service are external modules to the system.
g. The suitable resource selector sorts the resource(s) according to their number of freely available processing elements, and this is based on the resource information gathered from the distributed resource information service.

h. The distributed resource gateway propagates all gathered information to the schedule controller.

i. The schedule controller keeps record of all the sorted lists of resources and jobs that come from both the suitable resource selector and schedulers.

j. The transaction status is copied to the object base store and saved by the schedule controller. (This comprises of generated schedule, selected resource information, and also the scheduler process state). Subsequent scheduling events are updated via the same update process.

k. The matchmaker receives copies of lists of sorted jobs and resources transferred from the schedule controller, for possible consideration of resource allocation and execution. The transaction status is subsequently copied and saved to the object base store. If allocation is considered possible based on the adopted policies, then step 12 follows, otherwise, the matchmaker send queries to the schedule controller for optional remote cluster resource(s).

l. The matchmaker finally creates one-to-one logical mapping between the two sorted lists (job and resource lists).

m. The generated mapping lists are transferred to the distributed resource management system for execution on the local cluster(s). Subsequently, the distributed resource management system sends information about the results of execution (successful or failure) back to the distributed resource gateway for the application user consideration.

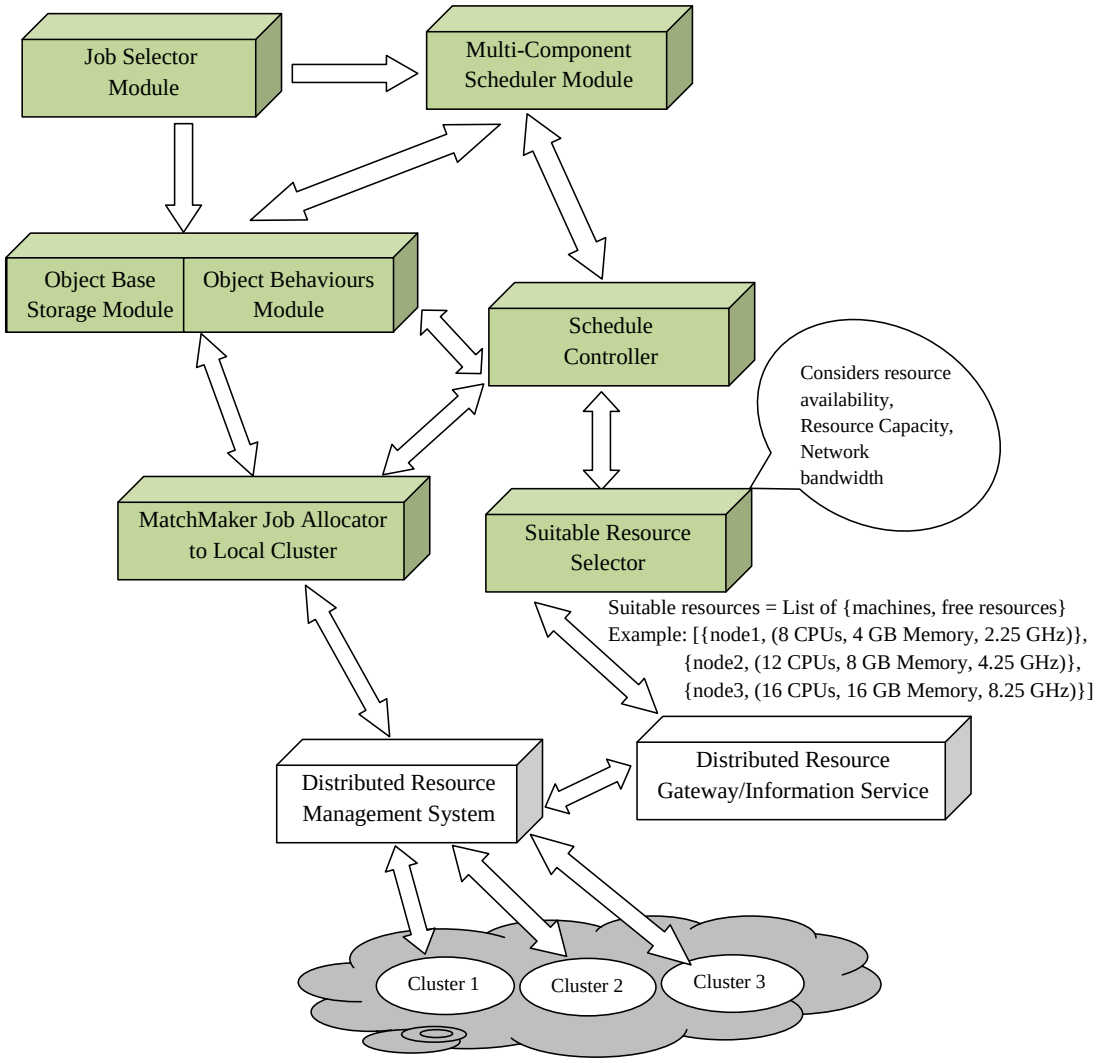


Fig. 3.14b: Multi-Component Job Scheduling

I. Some basic assumptions made for the proposed scheduling model

a. Job resource(s) requirement must be specified for those jobs that need resource(s) from multi-clusters.

b. Resource description (attributes) should be known to the scheduler, examples include; cluster id, cluster name, number of machines, number of processors on one machine, processor speed, and so on.

c. Job resource requirement description should be rigid (that is, requires a fixed number of clusters, fixed number of processors).

d. Resources in this case, refer to the number of available processors, processor speed, main memory, Network Bandwidth, and storage capacity.

e. Job in this case, implies parallel data-intensive or compute-intensive applications with or without interactive tasks.

f. There is no pre-emption, and all nodes allocated to a job are only released simultaneously when all of its tasks have finished.

g. Jobs may be split into smaller units called tasks. The definition of execution modes may also include modes for job splitting.

h. Resource may have some associated performance functions that can be used to evaluate it performance (resource configuration attributes, such as, processor speed, memory size, associated bandwidths, and so on).

i. Only simultaneous resource possessions of the jobs are considered.

Fig. 3.15 presents and describes the main components of the Object-Oriented scheduling framework described in this section. The first illustration is an example framework that describes the complete model operation of the single-component scheduling framework (Fig. 3.15a), while, the second example describes the processes involved in the multi-component scheduling framework. These figurative descriptions follow from the earlier detailed explanation of the intended scheduling design patterns meant to proffer solution to the limitation in design paradigm identified in the existing scheduling systems.

In addendum, the component-scheduler forms the core component of the new scheduling system model which plays the role of a super-scheduler. Its main activities are resource selection, mapping and scheduling. The resource selection is the process of selecting the most qualified candidate resources which are available and fulfils user application requirements. The mapping process performs the function of both assigning

application tasks to the selected compute nodes and distributing data across those nodes. Finally, the scheduling process involves the allocation of computation-over-time.

J. Proposed Scheduling Simulation Scenario

Examples of the single-component and multi-component scheduling scenario are illustrated in Fig.3.15a and Fig.3.15b. The scheduling process starts with the agents' job submitter. The work of this agent, is to analyse the selected jobs and classify them into single component jobs and multi component jobs based on their resource requirement. It calls the appropriate scheduling algorithm (single-component or multi-component scheduler) for allocating resources to the selected job. The scheduling process of the former entails restricting the scheduling of user applications on resources from single cluster.

In a single component-scheduling process, when a user submits an application for execution, the application is profiled. The result of the profiled description information is forwarded to the scheduler (step 1, in *Fig.3.15a* and *3.15b*) as an input. The scheduler agents' on acquiring the detailed knowledge of the application resource requirements, proceed to search for the most suitable resource and when discovered, selects the resource for the application (step 2). Once the selection is made, the matchmaker agent maps each jobs with the selected resource nodes (step 3). Finally, the job is allocated to the selected nodes for execution. The thick dotted black lines indicate active nodes in the case of tasks executors. While for the scheduler, it indicates the three major roles performed by the schedulers during application scheduling and execution. These include; resource selection, job to resource matches and allocation of jobs to the selected resources.

In the case of the multi-component scheduling process (that is, when the value of $C > 1$), user applications require resources from multiple clusters. The same process of scheduling follows, as in the case of the single-component scheduling (that is for $C = 1$). However, the major difference between the two scheduling processes lies in the scheduler being able to initiate inter cluster interaction so as to select, map and allocate the best suitable resources across diverse distributed clusters for any user application.

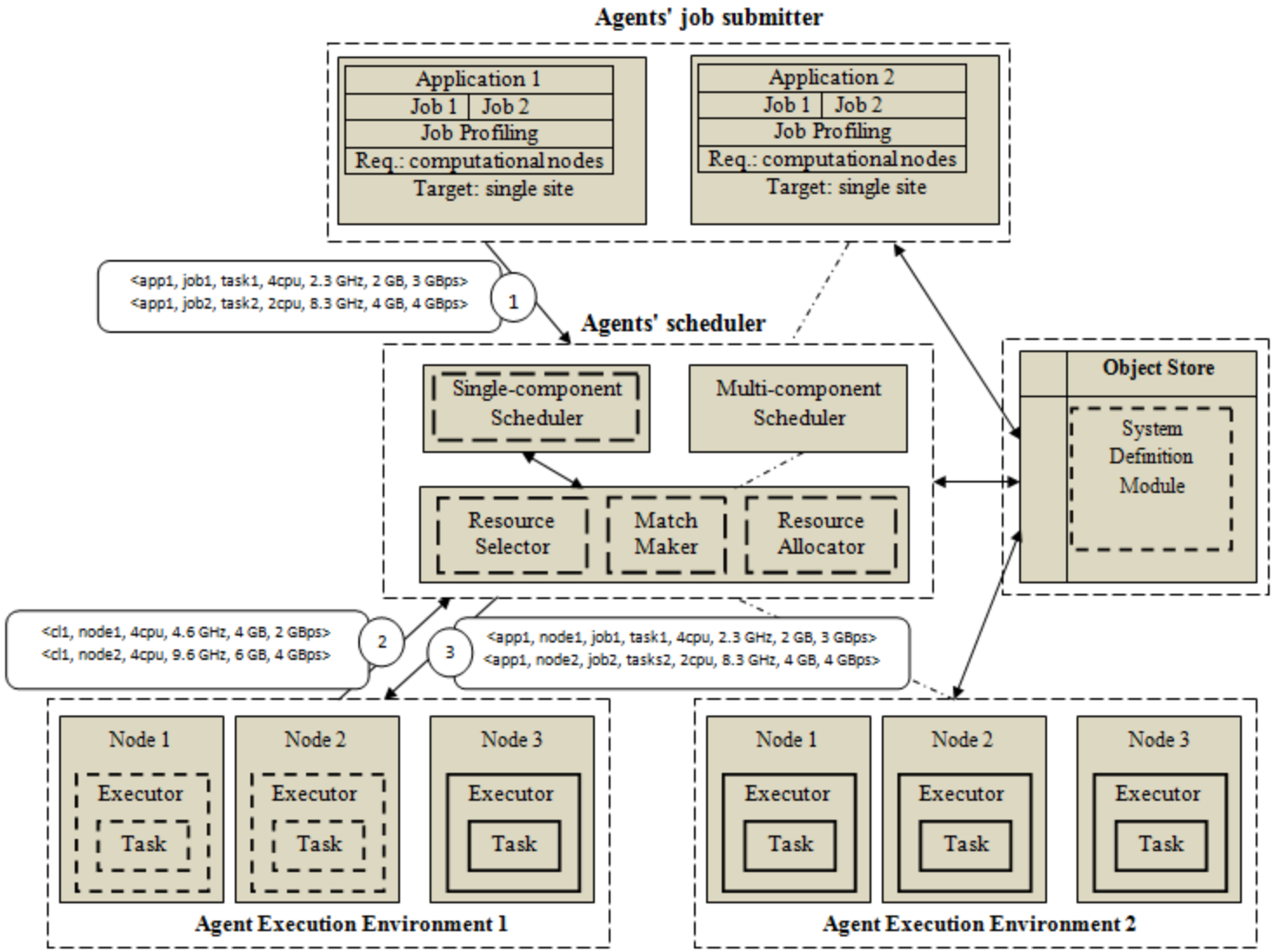


Fig. 3.15a: Single-Component Application scheduling Architectural Pattern

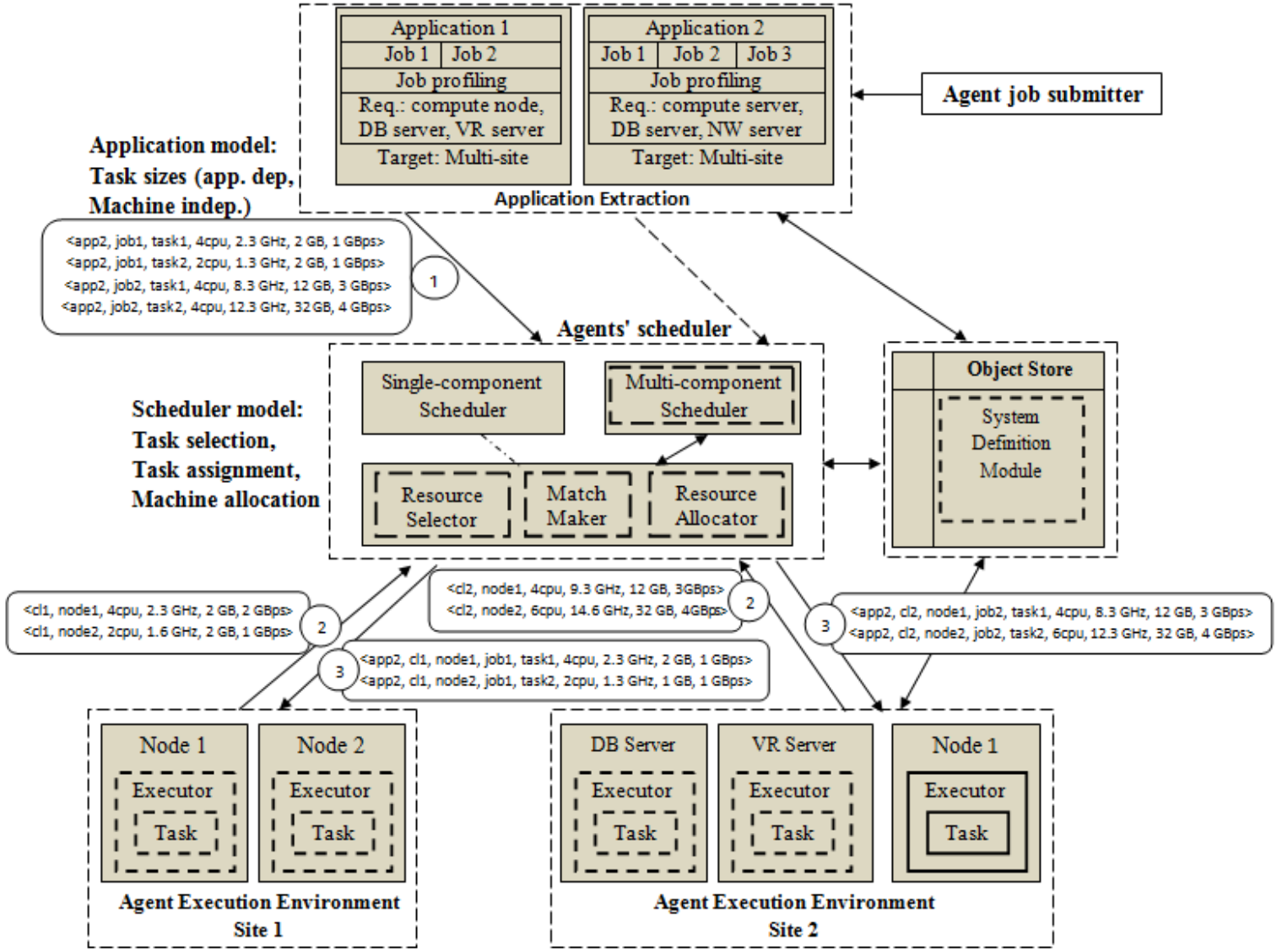


Fig. 3.15b: Multi-Component Application Scheduling Architectural Pattern

3.4.7 Multi-component Application Model

The underlying network of the distributed system contains a collection of heterogeneous sites connected by wide-area networks (Fig. 3.16). A site S is defined as a remote execution location, where an application can be executed based on the site resource capability. Therefore, each site offers an amount of one or more resources (with each performance evaluation based on the following; CPU speed, memory size, associated bandwidth, storage size, certain number of processors, and so on) and have a point-to-point bandwidth connection to each site resources ($bw_{i,j}$), that may differ in their connectivity strength. Resource capability has been determined empirically, using the mathematical model presented later in Section 4.2.2 and 4.3.4.

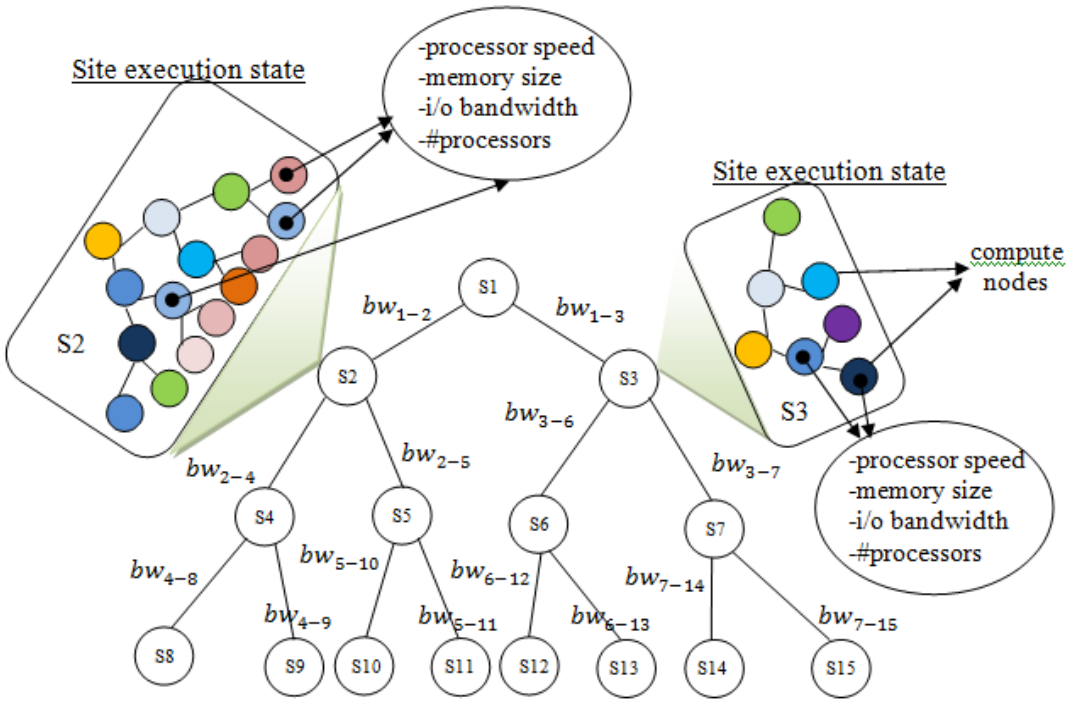


Fig. 3.16: Distributed System Network Model

Multi-component applications consist of a set of distinct application components that may communicate and interact over the course of the application execution period

(Weissman, 2000). Generally, component in this case, can mean sequential or parallel schedulable computations, which resource requirement include; remote application servers, high performance computing servers, specialized remote instruments, remote databases and so on. In other words, components as used in this dissertation refer to, set of resource components that an application requires for its execution (Fig. 3.4.17).

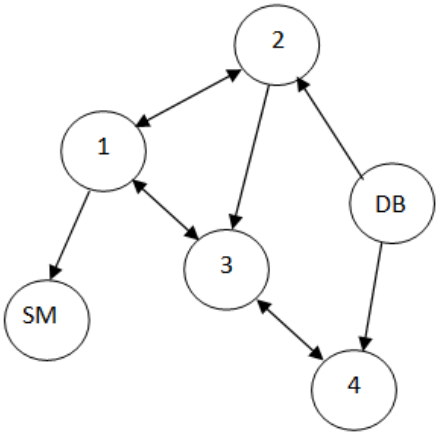


Fig. 3.17: Multi-component Application Model

Some application resource requirement components can be fixed and do not require scheduling. An instance of this type of component is the remote database server. However, the non-schedulable components may have the tendencies of impacting the scheduling of other components. For example, the placement of components 2 and 4 in Fig. 3.17, may be influenced by the amount of data transmitted by a database to 2 and 4. If a great deal of data is moved, then a high-speed link between 2, 4 and the database may be required. It is also possible that other components are fixed due to scheduling constraints, such as a given program component must be restricted to run in a particular site.

In the proposed architecture, the resource model, and specifically, machine model, can be set using two parameters $\mathbb{C}_1$ and $\mathbb{C}_2$. The parameter $\mathbb{C}_1$ indicates machine types,

while the parameter $\mathbb{C}_2$ points out the number of machines in the machine environment. If $\mathbb{C} \in \{P, Q, R\}$ as previously described in section 2.1.4, then three categories of machines can be identified, which includes identical parallel machines, uniform parallel machines and unrelated parallel machines, each with their respective processing time p_{ij} of job $j_i \in J$ on machine $m_j \in M$, defined as follows.

$$p_{ij} = \begin{cases} p_{ij} = p_i \; if \;\; \mathbb{C} = P \\ p_{ij} = p_i / s_j \;\; if \; \mathbb{C} = Q \\ p_{ij} = p_i / s_{ij} \; if \; \mathbb{C} = R \end{cases} \qquad (3.1)$$

3.5 Objects Interaction and Collaboration Pattern

In accordance with object-oriented paradigm, a design pattern is a solution to a general design problem in the form of a set of interacting classes that have to be customized to create a specific design. The proposed work revolves around the utilization of the interaction and cooperation that could be established among objects and object colleagues in the quest to solving distributed problems. In line with this aim, the object relationships is modelled based on collaboration pattern that leads to proactive, autonomous behaviour where a user's direct intervention is not necessarily required. The distributed collaboration pattern for object-oriented system is utilized here in order to avoid any kind of communication bottleneck that might arise in the new scheduling platform. In this case, each participating object type would require an instance of an object behaviour class, which is declared and referenced as object class of *class* (or nested class) for every interface that it supports, and it would have to maintain a library of information or knowledge of every other object it would want to interact with.

The object behaviour class controls access to the real subject, and can also provide a distinct interface. The job object, schedule object and resource object subscribe to the job interface, schedule interface and resource interface, while, the object behaviour

class:class supports the object colleague interfaces. A group of colleague objects would have similar interface implementation at every level of interaction. For example, all schedule colleagues have the same interface implementation as long as they possess the same behaviours and functions at that same level of interaction, which differs from all other object colleagues at some other levels. Another function of the object behaviour class is that of instantiating object types, object states and object behaviours, this class similarly defines all the basic characteristics of colleague objects. Object functionalities with their respective linked roles are as well defined in this same module. Therefore, communication that leads to interactions and coordination among objects and object components are initiated via the object behaviour class, which is a nested class on its own.

Defining communication protocols is essential for the object coordination. However, for object collaboration through coordination protocols, the object must be able to determine its behaviour based upon the state of the coordination protocol that it is engaged in. One object may be engaged in several communications simultaneously, requiring that it be able to accomplish context switching. The Object pattern externalizes an object's internal state so that the object can be restored to this state later. The object class that support collaboration and coordination which involve interaction among colleague objects must store and recover their states, which is very important for robustness in terms of system failures. With the object pattern, they can delegate this to a colleague object. The class is parameterized by protocol, which gives the possible states and transitions. The resulting pattern for the decentralized object-agent collaboration is shown in Fig. 3.18.

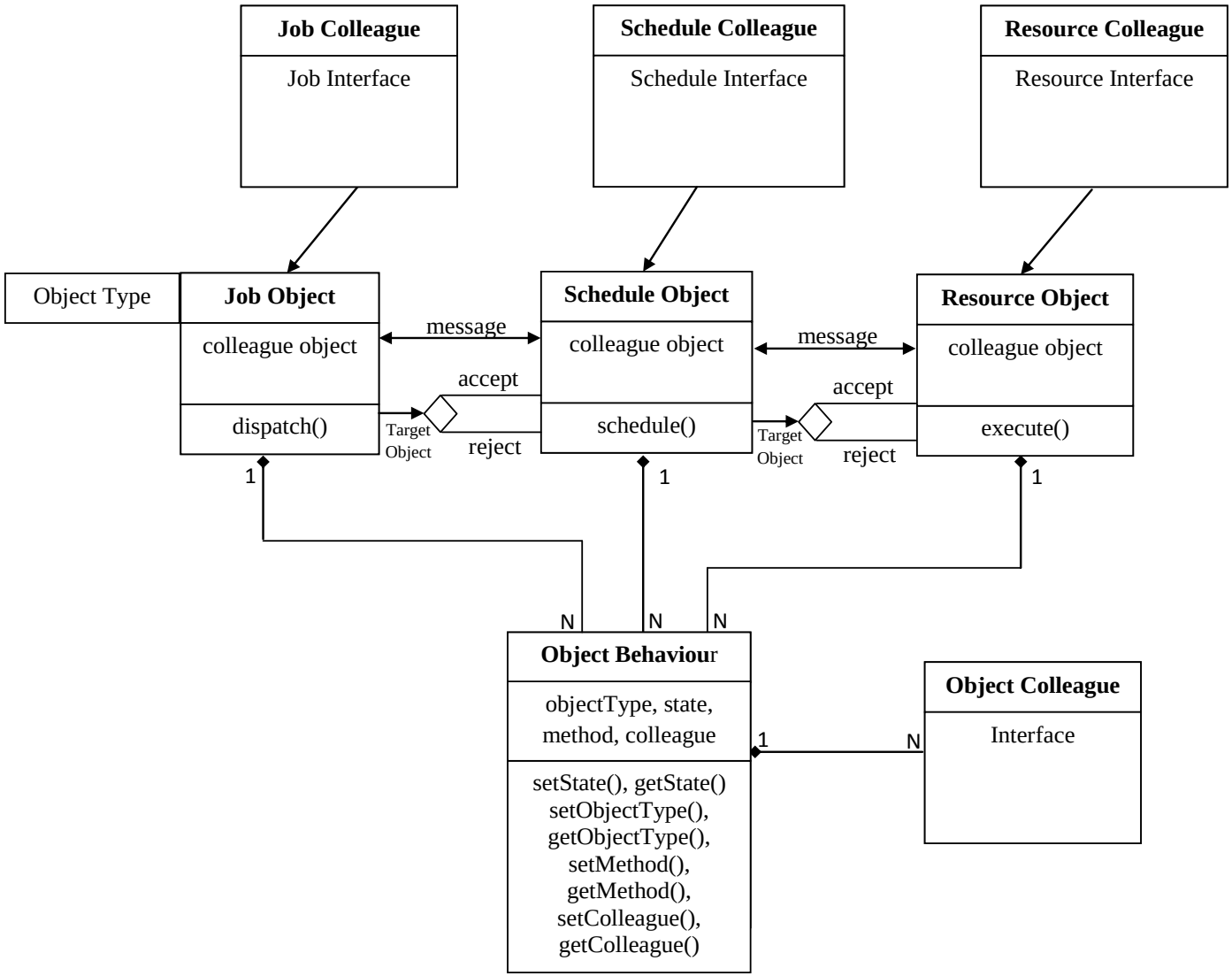


Fig. 3.18: A Pattern for Decentralized Object Collaboration

3.6 The Proposed Multi-agent Scheduling Model

In this section, the distributed multi-agent system architecture is presented. The aim is to develop a unified object-oriented multi-agent scheduling system by marrying these two technological concepts together (i.e. object-oriented technology and multi-agent system). By representing each important scheduling system objects (jobs, scheduler, machines) in the distributed scheduling system as an autonomous intelligent agent or a group of intelligent agents, multi agent modelling of distributed scheduling system is designed. A schematic diagram of the proposed system is shown in Fig. 3.19.

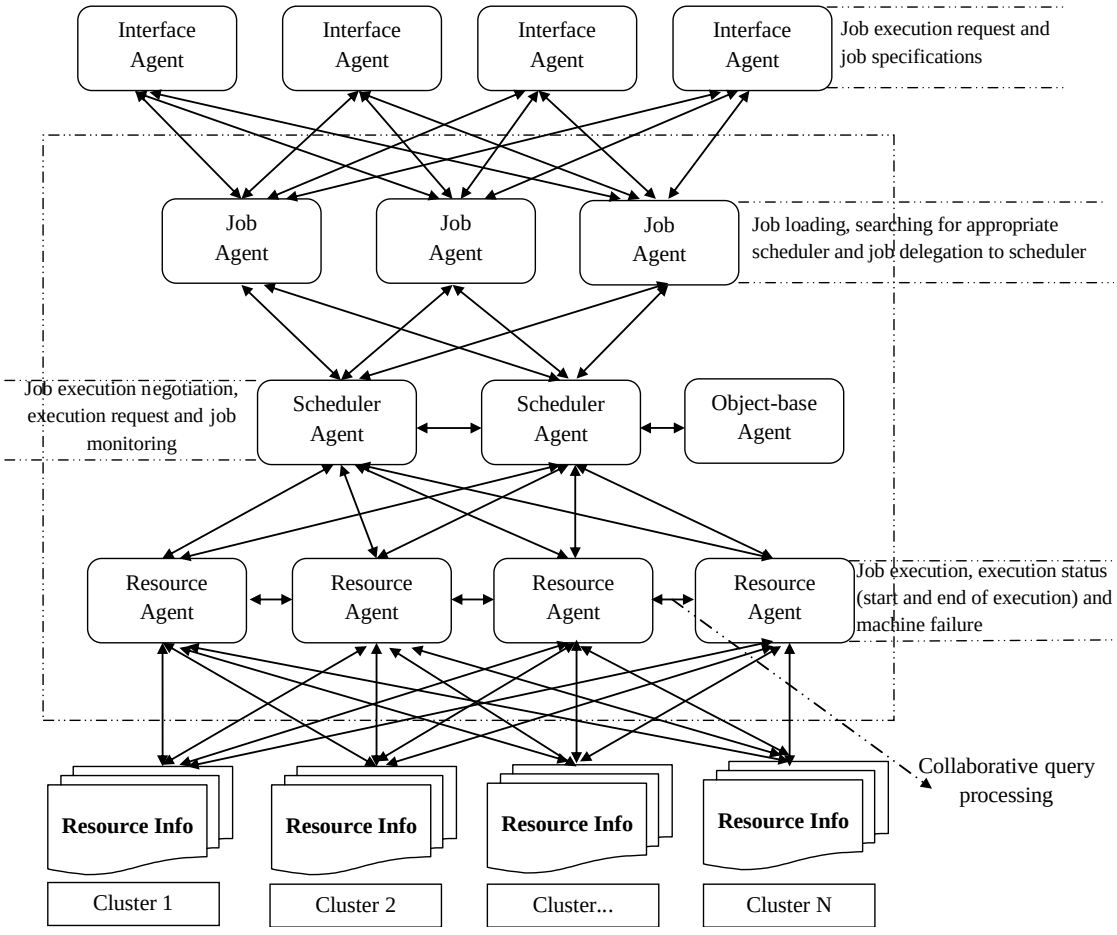


Fig. 3.19: Multi-Agent Architecture of the Proposed Scheduling System

The multi-agent based scheduling model illustrated in Fig. 3.19, has five types of agents. Interface agents that interact with user's job. Job agents that load and analyse

users job, by separating the jobs into two job components (single-component and multi-component). The job agents also perform the task of searching for an appropriate scheduler and delegating jobs to the discovered scheduler. The scheduler agents perform the task of job scheduling by formulating scheduling plans and carrying out these plans through querying and exchanging information with the resource agents. Object-based agent is mainly used for agent knowledge representation. Resource agents provide intelligent access to heterogeneous collection of local resource sources.

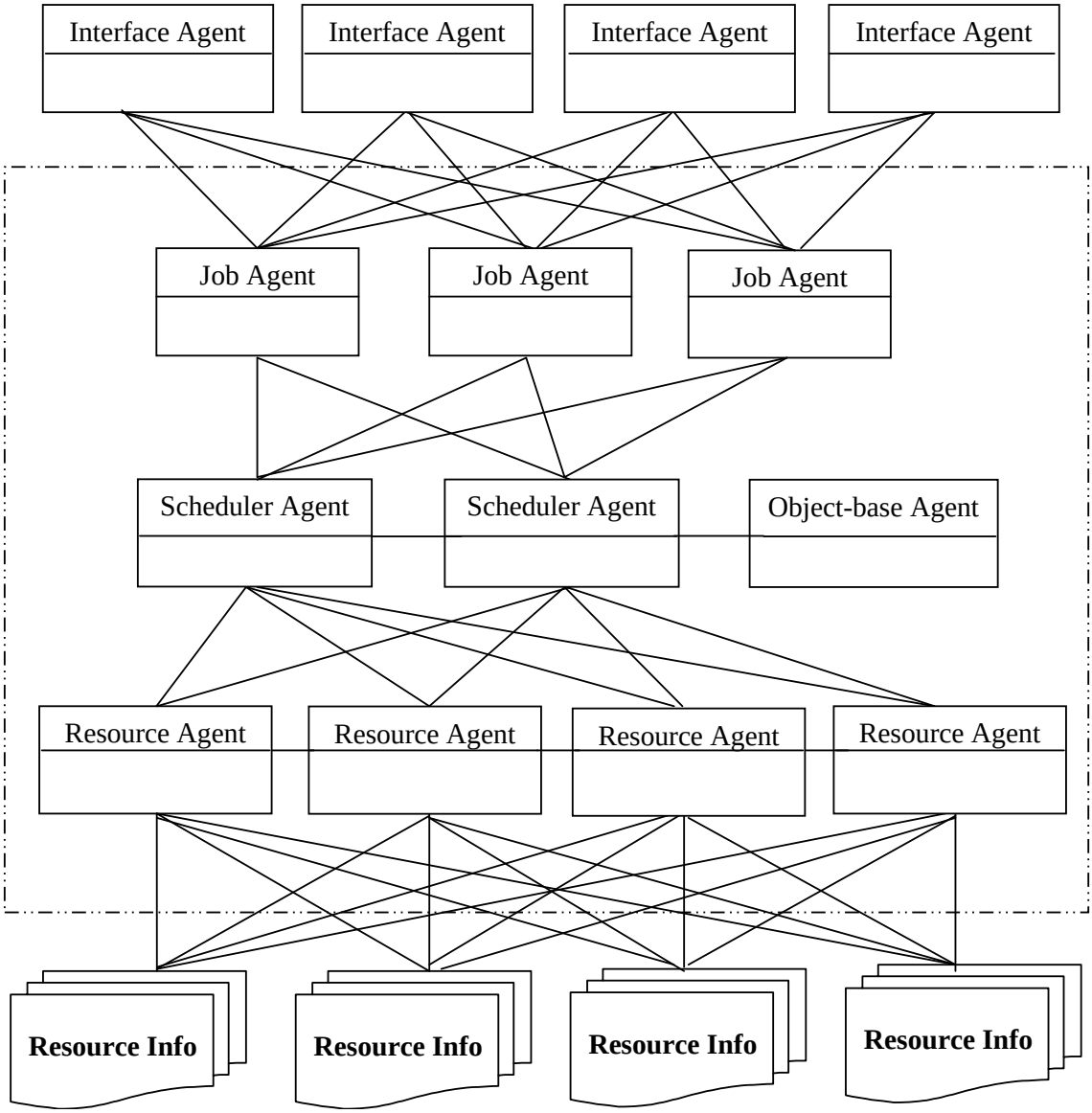


Fig. 3.20: Object-oriented Multi-Agent Scheduling System

There are some limitations in following the traditional object-oriented design techniques in building a multi-agent system. These limitations include, representing collaboration among agents and agent mental states. To resolve these problems, the architecture centric design methodology for multi-agent systems was suggested by (Yim *et al.,* 2000). One benefit of this method is that, it helps to constrain modelling. The concept is based on considering multi-agent system as a software system. It uses object-oriented methods as a core development methodology, and extends them to properly represent collaborations among agents. This approach is used in this dissertation to construct the object-oriented agent-based scheduling system.

A typical example of the agent's mental state representation is covered in the design of the object oriented database storage model, and it can be used to describe and represent the various agents' knowledge, which intent is to facilitate knowledge base maintenance.

3.6.1 General Scheduling Blueprint for Multi-Agent System

This section of the work introduces the system blueprint that would be used to develop the proposed object-oriented scheduling system platform. In line with this initial conception, there are three major parts to the model presented in Fig. 3.21, the client jobs in activation queue, the distributed scheduler, which is made up of autonomous agents, and the distributed computational resource environment that house the computational resources. The focus of this dissertation lies on developing a multi-agent scheduler that is based on the concept of object-oriented patterns. However, the main focus is to model and implement a scheduling system whose purpose and processes are intended to solve parallel job scheduling problem by matching client jobs to preferred available resources in the distributed computing environment. Again, this scheduling process is strictly based on the adoption of object-oriented techniques that encompasses

the utilization of various object entities, attributes and methods applied to the participating object components. Different object entities communicate among one another through the process of object method calls.

Everything relates to about objects, ranging from user submitted jobs, agents and resources (machines). However, agents in this case, denote computational nodes with processing abilities and some level of intelligence. Agents are further classified into three categories based on their functions. Job agents, this category of agents handle the processing of users submitted job by loading the job into the system and searching for the suitable scheduler to delegate the scheduling activity based on the job component types. The operation or method commonly applied to these set of agents is the load and delegate methods. The second class of agents are the schedule agents, this agent is responsible for generating schedule plan and assignment of tasks to best matched resources. The schedule and dispatch methods are commonly used to trigger events among these agents. The third category of agents, are those used for parallel job execution and generation of schedule results. The executor and kill commands are the two basic operations used to initiate the process of job execution.

It is assumed that every object by default, is an agent and vice versa. And each of the object or agent has a defined object class name, methods or set of operations that are applied on them and set of attributes that make them unique from other non-colleague objects or agents. Due to the similarity between object and agent role, in the context of this dissertation, these two words would be used interchangeably.

Our intent generally, is to develop a complete Grid scheduling framework. A complete Grid scheduling framework would normally consist of four building blocks; these include the *application model*, the *resource model*, the *performance model* and the

scheduling model (policy). An *application model* extracts the characteristics of the users applications to be scheduled. The *resource model* provides a fine-grained description of the underlying distributed system resource characteristics (example, processing speed, main memory size, storage capacity, link bandwidth, etc.). A *performance model* is responsible for computing and predicting the performance potential of a particular schedule (example is the computation of goodness function or quality of available resources with respect to a specific user application resource requirements). The *scheduling model* is responsible for schedule planning and generation; this model decides how user applications should be executed and how resource should be utilized.

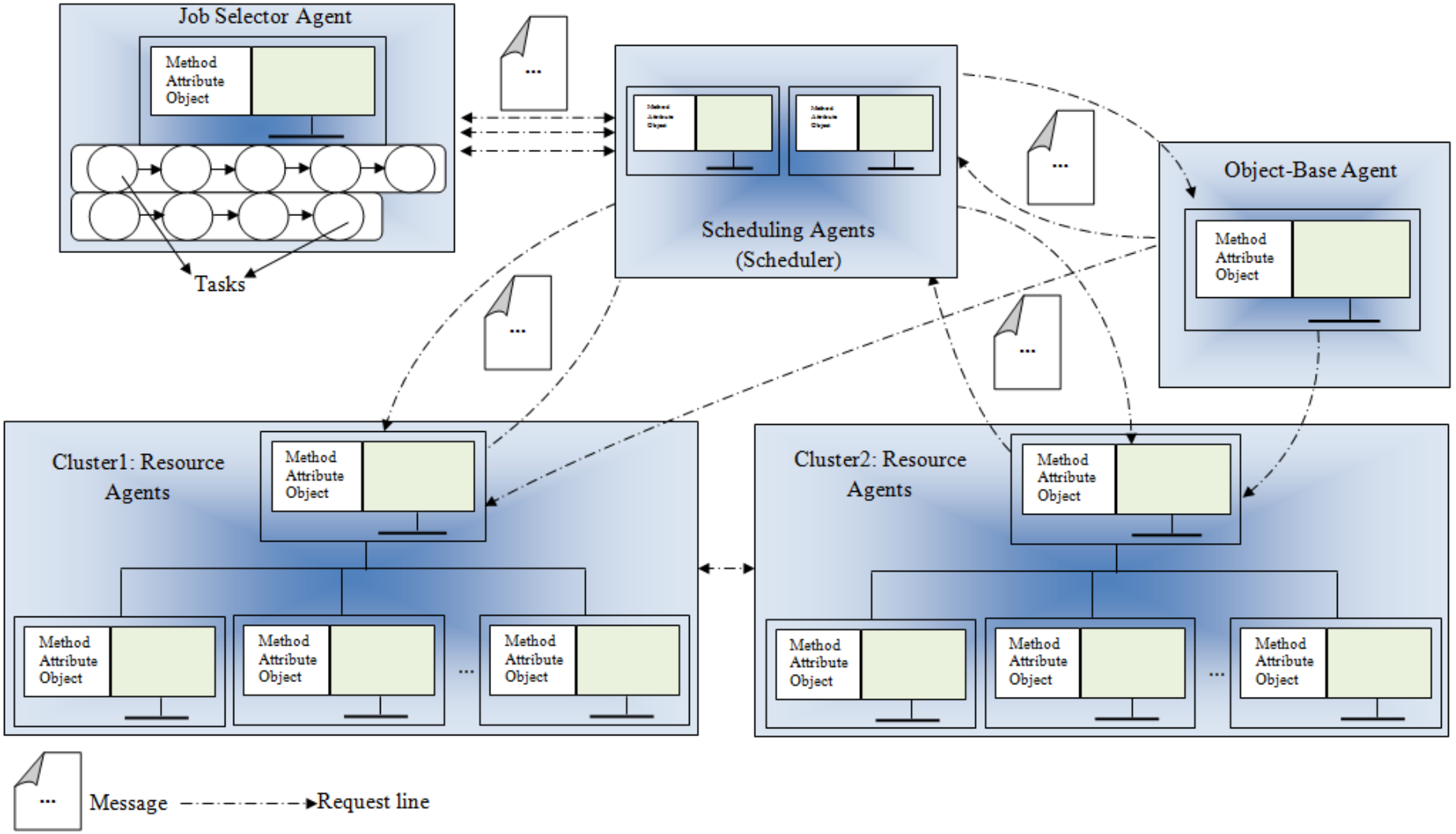


Fig. 3.21: Object-Oriented Scheduling System Blueprint

The Fig. 3.21 depicts an initial conceptual model of the object oriented multi-agent scheduling system. Here, a claim is made that jobs could be regarded as an object, while, a task is an instance of the job object. It is also assumed that each instance of a job has an attribute and a method attached to it as shown in each block represented in the Fig. 3.21. Resource is another object entity with unique attributes and methods. Same goes to the distributed multi-agent scheduler. Autonomous agents act on behalf of clients' jobs by generating an executable schedule plan and dynamically discovering and matching jobs to computational resources using job descriptions and resource state information. Interaction between agents and resource is based on common relationships of objects, object attributes and object method invocation (the is-a relationship and has-a relationship type). It has been mentioned earlier on that the object-oriented system are usually designed around two basic entities, namely, objects and methods.

There are two basic types of methods that can be used to trigger an operation or initiate an event in an object, particularly those associated with sets or lists (or arrays) and graphs. In general, the concept of primitive and functional methods is well assumed as long as the object oriented paradigm is concerned. The reflection of this concept is best explained using data structure, and some of the concrete instances can be found in many data structure literatures. Part of the two methods can also be applied to group of objects. In general, in a class-based object-oriented language, state is carried by instances, methods are carried by classes, and inheritance is only of structure and behaviour. Data type definition is another significant aspect worth considering here.

The concept of data types definition can also be considered vital, as operations can likewise be applied on those data types to achieve desired results. The fundamental idea of object orientation is that, the most powerful way to organize a program is to define data types and then associate operations with those data types. In particular, you want to

be able to invoke an operation and have the exact behaviour determined by the type of the object or objects on which the operation was invoked.

Tables 3.1 and 3.2 are lists of the two basic categories of methods, that is, the primitive and functional methods. However, the primitive type of methods are the most basic, in that they can be applied to any kind of object (Table 3.1 presents some of this method types.

Table 3.1: List of Primitive Methods Applied to Objects with their Descriptions

Object Specific	**Description**
construct	Create an instance
destruct	Destroy an instance
Copy	Create a new instance identical to a given instance
Assign	Assign a value to an attribute
retrieve	Find an instance with a given attribute

Table 3.2 contains a list of some basic functional methods with their description that can be applied to defined objects that correspond to elements, sets, as well as graphs. These methods are used majorly in the code aspect of this dissertation to implement the proposed object-oriented scheduling system. The application of this methods are presented in Chapter 5.

Table 3.2: List of Basic Fuctional Methods Applied to Objects Corresponding to Elements, Sets, and Graphs.

Methods	Description
insert([list])	Add items to the Set::Object.
includes([list])	
has([list])	Return true if all the objects in list are members of the *Set::Object*. List may be empty, in which case true is always returned.
contains([list])	
member([item])	Like *includes*, but takes a single item to check and returns that item if the value is found, rather than just a true value.
element([item])	
remove([list])	Remove objects from a *Set::Object*.
delete([list])	
invert([list])	For each item in list, it either removes it or adds it to the set, so that a change is always made.
equal(set)	Returns a true value if set contains exactly the same members as the invocant.
not_equal(set)	Returns a false value if set contains exactly the same members as the invocant.
intersection([list])	Return a new *Set::Object* containing the intersection of the Set::Objects passed as arguments.
union([list])	Return a new Set::Object containing the union of the Set::Objects passed as arguments.
difference (set)	Return a new Set::Object containing the members of the first (invocant) set with the passed *Set::Objects'* elements removed.
unique (set)	Return a new *Set::Object* containing the members of all passed sets (including the invocant), with common elements removed. This will be the opposite (complement) of the intersection of the two sets.
compare(set)	returns one of: "*proper intersect*", "*proper subset*", "*proper superset*", "*equal*", and "*disjoint*"
subset(set)	Return true if this *Set::Object* is a subset of set.

Fig.3.22 depicts an instance of the proposed scheduling model, that is concerned with integrating multiple problem-solving perspectives or objectives when generating solutions to a scheduling problem.

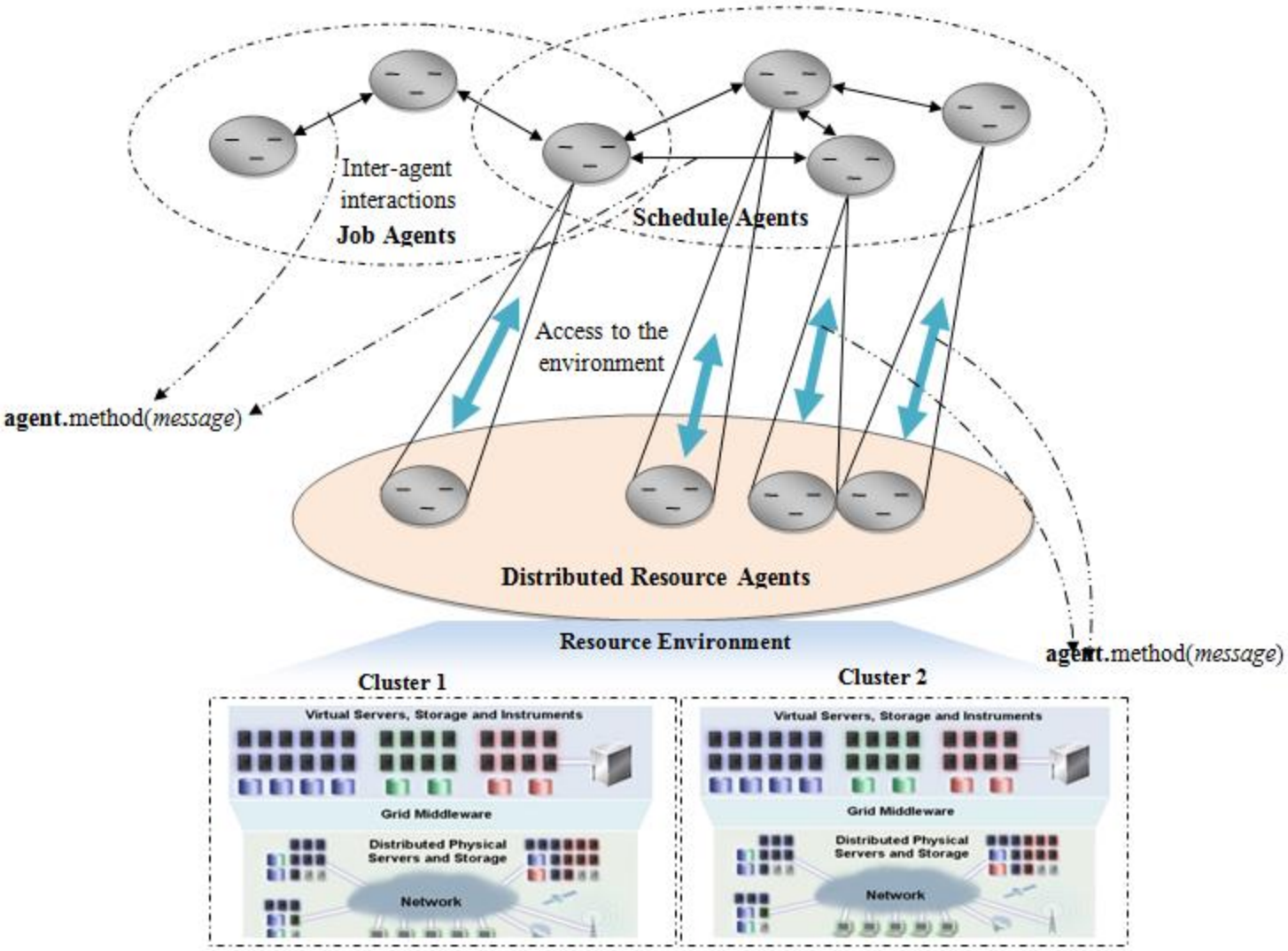


Fig. 3.22: Abstract Architecture of the Object-Oriented Multi-Agent Scheduling Platform

Let recall that, in distributed computing environment, large number of processes compete for resources (CPU, memory, storage, network) in order to execute their jobs. This in essence is typically a scheduling problem that is NP-Complete, and by this definition, we apply the same mechanism used in distributed problem solving domain to handle this issue. This approach involves solving the scheduling of distributed jobs consisting of several sub-tasks by sharing them on distributed multi-agent systems spread across peer-to-peer network. As stated earlier on, the principle of peer-to-peer computing is adopted and by so doing, the tasks which usually consist of programs, local memories, and collection of I/O ports uses the cumulative power and bandwidth among peers to solve their various objectives.

Since our target is a decentralized scheduling platform, the notion of client server systems is discarded. Decentralized processing is highly supported by peer-to-peer computing as all peers are assumed to be autonomous.

3.7 Agent-Based Communication Architecture

Figure 3.23 depicts a high-level overview of communication flow among the object-oriented multi-agent scheduling system entities, such as, Job Activation Queue, Network Links, Peer-to-Peer Routing, Multi-agent scheduler, and Distributed Resource Clusters, which are all instances of the object class. First, job is sent into the system from the Activation Queue that comprises of series of decomposed tasks in the form of scheduled plans. Schedule Plan here, signifies the definition of constraints and is represented here as precedence relations between tasks in the form of directed graphs explained earlier on. The tasks are sent out through the Activation queue output entity, which transforms the tasks into several packets based on the Maximum Transmission Unit (MTU) of a network Link. Tasks are delivered to the scheduler by means of peer-

to-peer routing. Therefore, two types of peer-to-peer routing can be distinguished, global and local routing. In global routing, a message is delivered close to the destination node that is outside the message originating neighbourhood node. In local routing, the destination is at a nearby neighbourhood node. By design, the proposed scheduling system aims at global routing (Fig. 3.23). When an appropriate resource node is found, then global routing happens, crossing a large part of the overlay and arriving to a nearby node. In chapter four, an algorithm and evaluation criterion for peer-to-peer routing mechanism based on the proposed scheduling application scenario is presented.

The decentralized scheduler entity performs the operation of task scheduling by assigning tasks to appropriate distributed resources. Scheduling here signify the assignment of resources to tasks at specific times. The scheduler which comprises of multiple intelligent/autonomous agents performs their scheduling activities based on the orientation of problem solving techniques. The network model adopted here follows the concept of Peer-to-Peer network. Once the schedule has been generated by the scheduler, the schedule sequence is routed (i.e. global routing) to the nearest best matched resource node for processing. For tasks to flow through the network links, first, there is a transformation of tasks into packet like data, after which they are then converted back when they arrive into the resource entity. This transformation thus, is required so as to enable tasks movement across the network and also to present them fit for computation.

The resource entity represents a resource with various properties such as number of processors, memory size, storage capacity, network bandwidth size and scheduling policies that must be met. Requests (or queries) are received by the resource nodes from the network link, and after execution, results are routed back to the appropriate

designation. The routing process propagates the data back to the requesting node. The distributed resources are modelled as object instances of some sort, which can either be some processing agents or computational nodes.

Operating within the context of distributed network environment, a task distribution mechanism is required to scatter the available tasks among the peer nodes. Some suggested algorithms for this purpose rely on a push strategy (Quinn, 2004), where peers with too many available tasks send some of them to neighbouring peers, while, other algorithms rely on a pull strategy, where peers with no work to do ask neighbouring peers for work. Both strategies will be implemented in this dissertation and will be discussed in details in chapter four.

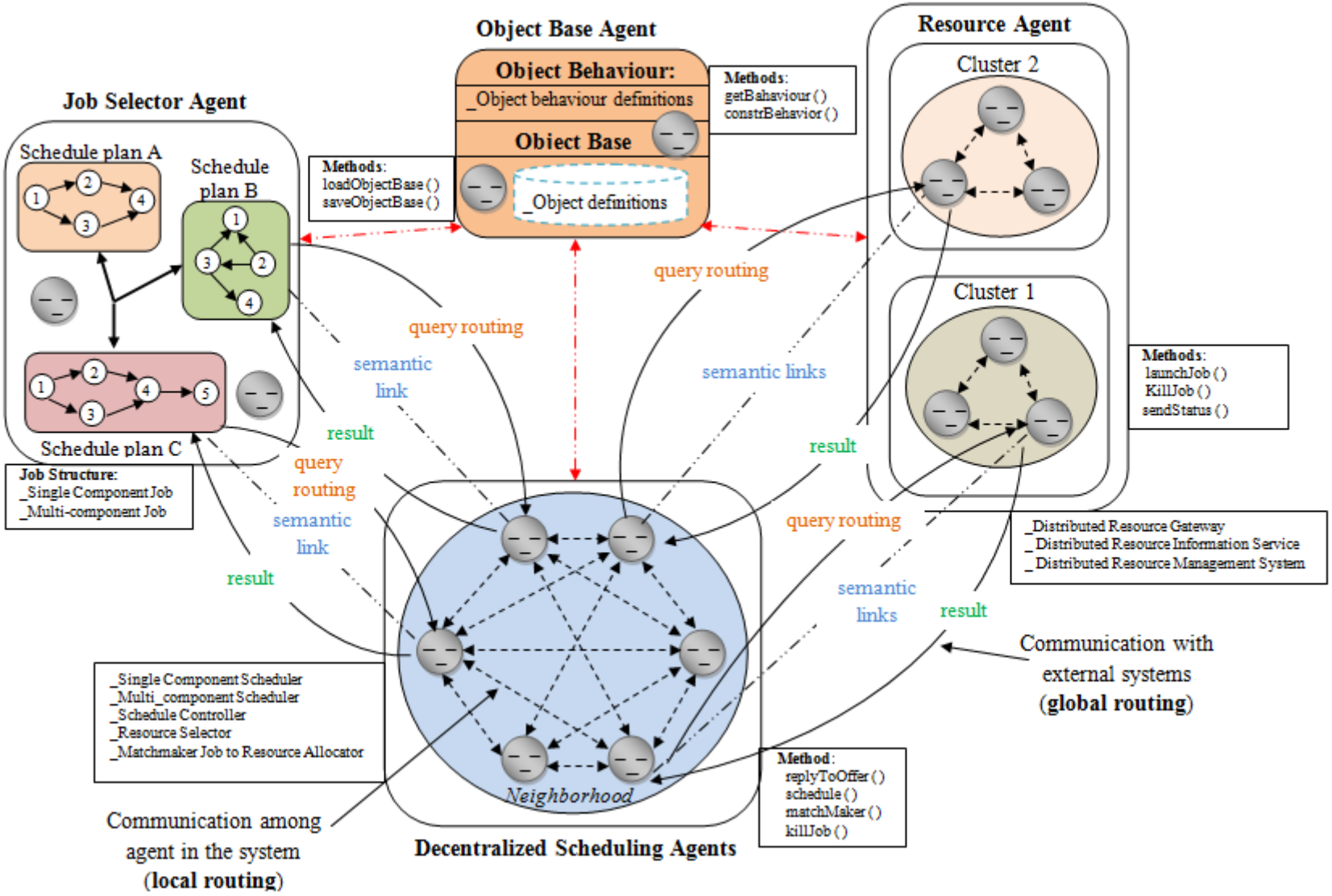


Fig. 3.23: Unified Prototype Architecture of the Object-Oriented Multi-Agent Scheduling System Framework.

A semantic link (a straight doted lines in Fig. 3.23) between two agent nodes represents a connection that denotes that one agent is related to the other in terms of similarity in behaviour, functionality or dependency of their skills or resources.

In Crespo and Garcia-Molina (2005), A Semantic Overlay Network within the context of peer-to-peer computing was expressed as an abstract layer that represents a set of semantic relationships established between agents that are randomly connected at a lower level of a peer-to-peer network on the basis of archieving common goals. Peer-to-peer overlay network can be grouped into two, structutred and unstructured.

An unstructured peer-to-peer network would usually consist of agent peers joining the network with some loose rules, without any prior knowledge of the topology. One typical example of the unstructured peer-to-peer overlay network is the Gnutella (Ripeanu, 2001). In the structured peer-to-peer overlay networks, network topology is tightly controlled and content is placed not at random peers but at specified locations that will make subsequent queries more efficient. The two commonly used structured peer-to-peer overlay networks include Distributed Hash Table (Naor and Wieder, 2003) and Content Addressable Network (Ratnasamy *et al.*, 2001). Fig. 3.24a represents typical example of structured peer-to-peer overlay network.

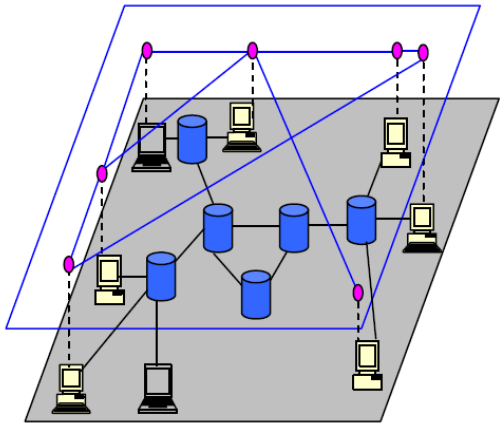


Fig. 3.24a: Example of Structured Overlay Peer-to-Peer Network

Basically, a semantic overlay network stucture represents the interconnections of semantically related network nodes. Fig. 3.24b represents the proposed multi-agent scheduling system defined on top of a peer-to-peer network topology. A semantic link (represented by arrow lines in Fig. 3.24b) between two or more agent nodes represents a network connection that denotes that one agent is related to the other in terms of similarity in operation or dependency of their skills or resources.

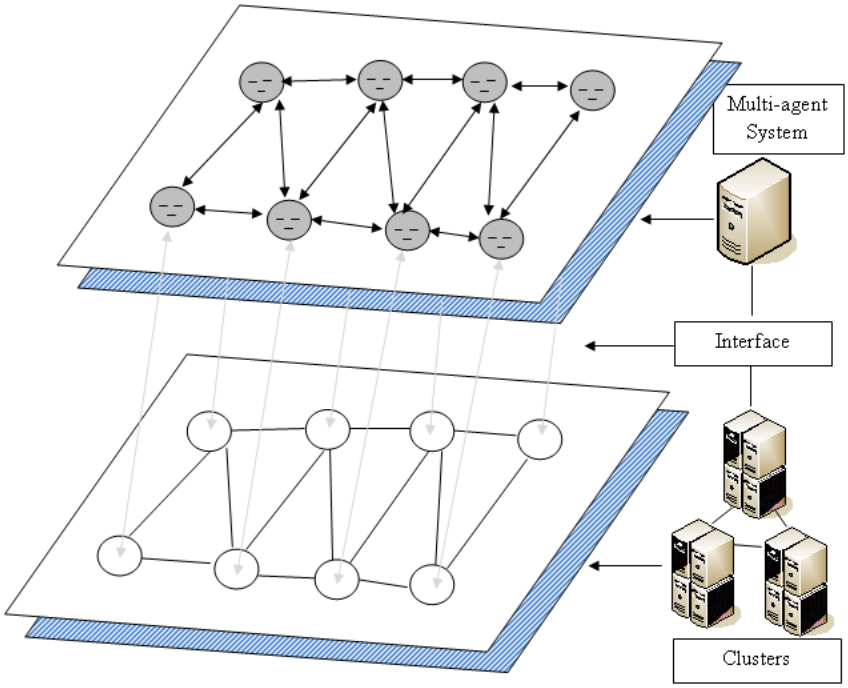


Fig. 3.24b: Semantic overlay network topology for the proposed peer-to-peer multi-agent system

The semantic network overlay in other words, can be seen as dynamic network structural arrangement which is designed to improve query performance while at the same time maintaining some high degree of node autonomy. The adavantage of this network organisation is that, agent nodes with semantically similar functionalities and intelligence are grouped together based on their common objectives.

Fig. 3.25 illustrates an instance of agent nodes connected by network links (represented by dotted lines). By applying semantic overlay network, agent nodes in block one perform the tasks of job analysis and job selection. Agent nodes in block two perform the task of scheduling loaded jobs from block one. Connection links are established between these agents. Similarly, the agent nodes in block three are assigned with the role of job execution, and since these jobs are dispatched from agent nodes in block two, they also establish connection links between these two blocks (scheduler agents and resource agents). However, agent nodes in block one have no direct connection with those in block three since it is mandatory for jobs to be scheduled first (in block two) before they are mapped to the appropriate agent nodes for exection (in block three). Fig. 3.25 shows a more simplified explanation of what is presented in section 3.4.3.

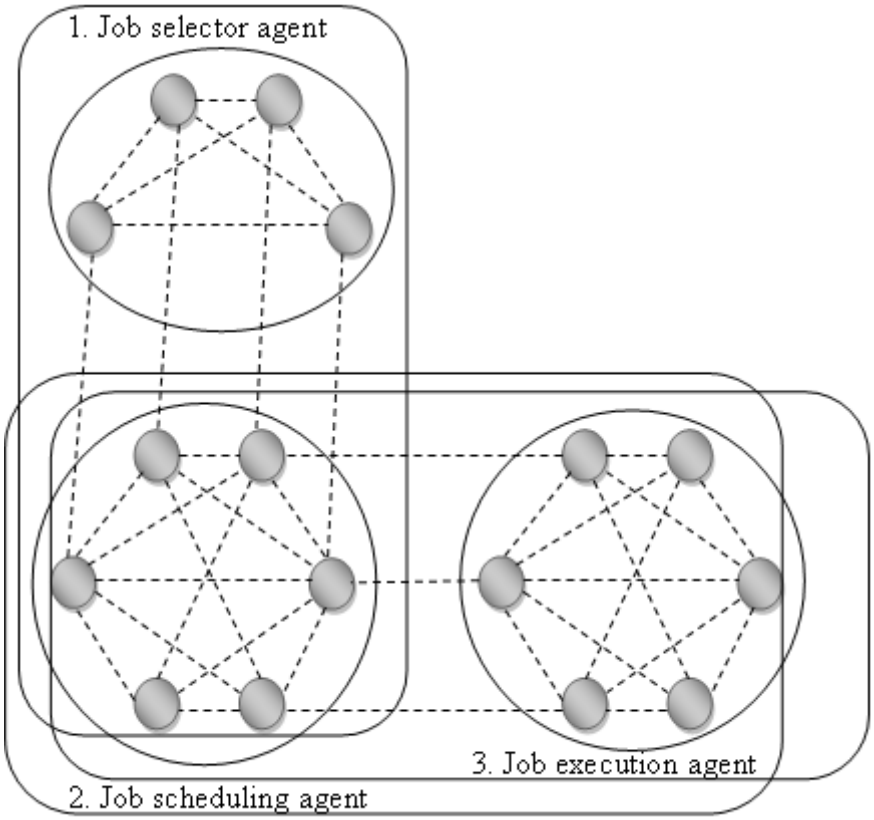


Fig. 3.25: Scheduling Network Model

3.8 Object-based Storage System Structure

The term ‘object base storage system’ in this dissertation, refers to a storage component part of the scheduling platform, which is one among the several components that make up the working system for a functional object-oriented scheduling system. The basic function of the object base storage component is basically to store definitions of all object types with their corresponding instances and methods applied to them. In other words, the object base storage system serves as library modules where schedule of object instances are defined together with the set of methods that triggers the actions that generate and execute schedule events. The advantages provided by this storage option are that, it makes scalable computing more feasible, reusable, manageable and effective.

Some of the commands set that are usually applied to object-based storage are as listed in Table 3.3:

Table 3.3: Object-Based Storage Command Sets

Object Specific	**Group**	**Others**
Create Object	Create Object Group	Set Attribute
Open Object	Remove Object Group	Get Attribute
Read Object		Flush
Write Object		Format
Append Object		
Close Object		
Remove Object		
Import Object		
Flush Object		

In (Hospodor *et al.*, 2002), the basic methods of object oriented architecture were given and are as shown in the table 3.4.

Table 3.4: Object-Based Storage Method Types

Methods	**Descriptions**
new storage = StorageObject(size)	Instantiate new Storage Objects
new moreStorage = StorageObject(size)	
metadata = storage.finger	Obtain metadata from a Storage Object
buffer = storage.read	Retrieve data from Storage Object into memory
storage.write = buffer	Store data from memory onto Storage Object
storage.append = buffer	Extend Storage Object with additional data
storage.find(regExp)	Search a Storage Object
sortedStorage = storage.sort(keyValue)	Sort a Storage Object by key value pair
moreStorage = storage.replicate	Replicate a Storage Object

Just as in the traditional database where data types are defined using the concept of data schema, in the same vein, object based data types and their corresponding instances can be defined as follows: consider an object type J, having unique attributes describing the features of the job with some group of methods applied on the job. If the following jobs $J_1, J_2, \ldots, J_k$ are instances of the job J, the same attributes and methods associated with J will equally be applied to the k^{th} instance.

Attributes play an important role with respect to object base storage system. It improves data sharing by allowing storage applications to share common sets of information that describe data, and examples are, access time, storage size, resource requirement and user information. All these can be efficiently stored with the object data and, in certain cases, interpreted by the storage device. Attributes also serve the purpose of creating awareness to storage devices on how objects are being accessed (Mesnier *et al.*, 2003). In most deployments, attributes will usually contain information analogous to that contained in an index node (node), the primary data structure used in many UNIX file systems.

3.9 Techniques for Achieving Components Reusability and Portability

As part of the proposed research objective, the target therefore, is to be able to design and build scheduling system components that would be reusable and portable to a vast number of distributed computing environments. The primary essence of software reusability is to shorten software development process. One other reason for software reuse is to reduce the time and cost of maintaining a software product. This concept of reusability has been achieved by following strictly the object-oriented design concept throughout the design stage in the proposed work.

One major solution to achieving portability for a particular piece of software that is deemed very important and with significant prospect for industrial utilization is to forbid programmers to use constructs that might cause problems when ported to another computer platform. A typical example would be to follow an obvious principle of implementing all software in a standard version of a high-level programming language, such as, Java and C++. Combining the above principle with some level of abstraction in object implementation could also increase some high level of portability.

The upper level, implemented in a high-level language, interprets the user's command and calls the appropriate lower-level code artefact to execute that command. If the graphical display routines are ported to a new type of workstation, then no change is required to be made at the user's code or the upper level of the graphical display routines. However, the lower-level code artefacts of the routines have to be re-implemented, because they interface with the actual hardware, and the hardware of the new workstation is different from that of the workstation on which the package was previously implemented.

3.10 Summary of Chapter's Contributions

This chapter proposed the design of a general-purpose scheduling system framework, which is based on both the concept of Object-oriented and multi-agents design patterns. A description of the proposed system architecture has been presented and discussed. The goal of this design was to define objects for every aspect of the scheduling system. As such, it promotes an extremely modularized design pattern which makes the system easily reconfigurable and extendable.

In order to achieve some of the research objectives of having a scheduling system that is free from those restrictions and limitations that come with the classical software implementation techniques, the design consideration presented in this chapter is used in conjunction with the multi-agent scheduling approach. The marriage of these two scheduling approaches helps to resolve the problem of both design and implementation issues pointed in the research problem statement, by harnessing the benefits that come with both the object-oriented design paradigm and the multi-agent technology.

A follow up in this direction of research is the development of modularized object-based high-level programming software application specifically designed for the development of scheduling systems. The advantages of such approach are obvious. It would significantly shorten the development and prototyping time. It would also enable easier extensions and modifications of the system by individuals with less knowledge of the inner details on the system implementation. The rest of the chapters of this dissertation will be dedicated to achieving this research direction.

CHAPTER FOUR

MULTI-AGENTS SCHEDULING COORDINATION, RESOURCE SELECTION AND ALLOCATION

4.1 Synopsis

One of the main targets in this chapter is to deploy the scheduling model designed in the previous chapter inside MAS. Dealing with scheduling problems within the context of distributed resources owned by multi-cluster providers emphasises the importance of autonomy. Many providers need to maintain their autonomy by keeping their own scheduling, security, and negotiation policies. Therefore, under these complex dynamics involving multi-cluster providers, autonomous behaviours are essential to meet up with the challenges. As such, proposal is made to address this aspect of the problem using MAS technological approach.

Dynamism with site autonomy can be handled with MASs. They rely on autonomous entities called agents to interact and make decisions based on internal logic (Frincu, 2011; Weiss, 1999). Multi-agent systems offer a natural extension to most RMS as they allow multi-providers to inter-operate through coordination and negotiation.

There are numbers of resource management platforms that use agents to handle configuration of running environment, transportation of jobs to multi-clusters local queues, executing user jobs and returning feedback in terms of results, to the client. Example of these platforms are: Condor (Frey, 2002), a distributed computing resource management environment for high-throughput applications which harnesses idle time on managed resources and has capabilities for their sharing too, it provides numerous advance functionalities such as job arrays and workflow support, check-pointing, job migration, rescheduling, and fault recovery, Condor-G (Imamagic, 2006) executes jobs

across remote distributed resource by using agents (grid middleware), and Task and Resource Allocation in a Computational Economy: TRACE (Fatima and Wooldridge, 2001), which dynamically allocates resources and agents based on the demand.

From the facts presented earlier on, it could be seen that, there are several of these agent-based RMSs platforms proposed to develop MASs for the purpose of job scheduling. However, most of them focus on individual aspects such as coordination, negotiation, scheduling, and feed-back control, with design implementation from the perspective of classical design approach. Providing completely distributed, scalable and customizable scheduling solutions for multi-cluster usage, is still a very big issue from both technical and scientific point of view. These **challenges** can arise from issues with **compatibility** when binding together software modules for distributed communication. Another factor can be attributed to the fact that some of the existing distributed scheduling systems lack **proper** planning with regards to **design and implementation** approach.

From a global perspective, a scheduling model which is capable of representing the system components as autonomous and fully customizable entities is required. It is based on the fact that, there is no current solution that addresses all the previously identified problems, that a new solution is being proposed to address some of the main challenges, by creating a scheduling system whose design principle and implementation approach is based on the application of object-oriented design concept and MAS scheduling platform.

Adhering strictly to the combined implementation methods of the object-oriented design principles and MAS, will be able to tackle both the technical and scientific challenges

associated with current scheduling systems. More so, it would also provide us with a scheduling platform that is able to:

a. Address design issues by reverting from classical design techniques to object-oriented design and implementation techniques.
b. Apply the intelligence and capability of autonomous MAS to distributed scheduling problems, rather than relying on algorithmic and heuristic solution-based approaches.
c. Offer fully distributed object-oriented storage and communication mechanism, that would serve as a yellow page for agent object's attributes publishing and interaction medium.
d. Offer coordination and cooperation mechanism among scheduling components/entities.
e. Support autonomy among providers by separating the dynamically changing scheduling policy from application, by means of components modularization. This invariably allows providers to maintain their own internal scheduling policies.

4.1.1 Multi-Agent Scheduling Platform

The choice of developing and implementing a distributed scheduling system using intelligent MAS, requires coupling of several individual intelligent agents together and ensuring their coordination and cooperation toward working together to tackle a given scheduling problem. In line with this choice, a new distributed scheduling system is proposed, where agents are classified and defined as scheduling objects or entities (section 3.5). The scheduling model comprises of three different intelligent agent layers (job processing agents, scheduling agents and resource agents). Each of these agent layers is enhanced with custom modules that support scalability and platform

interoperability, as number of agents can be increased to carter for daunting scheduling tasks.

In the work of Frincu (2011), agents were defined as intelligent control loops made up of single or several modules. Each stage of the control loop supports scheduling activities, by implementing the four key steps performed by an autonomic MAPE (Monitor, Analyser, Planner, and Executor) loop. However, in our model (section 3.3.3 and section 3.5), each agent module corresponds to one of the monitor, analyser, planner, and executor activities. For instance the analyser and planner activities are performed by the scheduler agent module, while the executor activity is performed by the resource agent module. As shown in Fig 4.1. The monitor activity however, is external to the scheduling activities, as it consists of lower-level modules, such as hardware dedicated scheduler, system monitoring, and gathering of scheduling information.

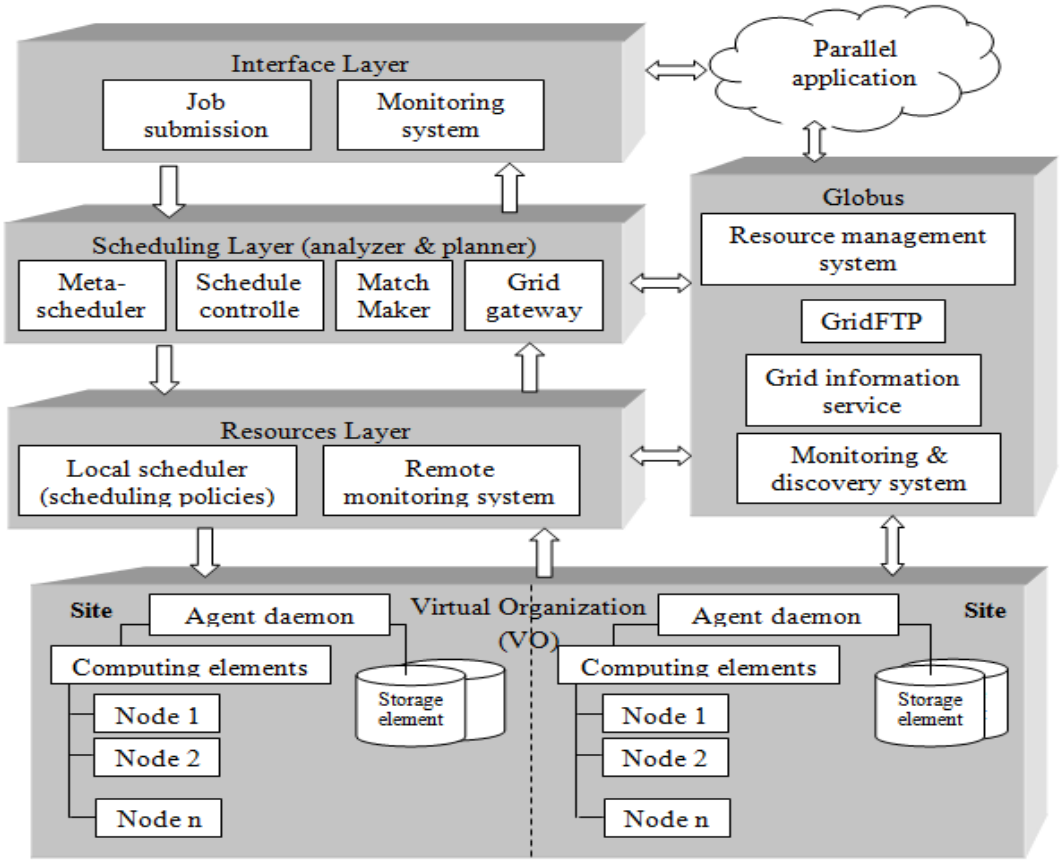


Fig. 4.1: MAS Scheduling Platform: A High Level Architecture

By adopting the concept of modularization, agents in the proposed model have been assigned to assume the role of all system functionalities. These agents' functionalities thus, include system environment monitoring, several of the activities involving job scheduling, resource allocation, inter-providers collaboration, job execution, etcetera. Most importantly, the modular structure of proposed platform, allows each provider the opportunity to maintain its own local scheduling autonomy as regard to scheduling policies. At the meta-scheduling level, agents are permitted to employ their respective scheduling policies, so as to enforce agent autonomy at different levels of system operation, by allowing them decide when to request or send tasks from or to other agents from different providers.

In summary, the agent-based scheduling solution is based mainly on agent collaboration, agents' constant evaluation of the condition of the system environment, and the decisions that agents make based on their assignment policies and beliefs. On this basis, the overall goal is to achieve the execution of the tasks in the shortest possible time, assigning tasks intelligently to the different execution sites of the distributed systems, so that the workload is distributed homogeneously between the sites. Each site should have the largest possible workload to decrease the waiting and delivery time of the tasks and reduce the total delay time.

4.1.2 MAS Communication Mechanism

The cooperation between distributed agents working together to tackling issues regarding scheduling of parallel jobs across multiple providers' sites, rely heavily on good communication strategies, likewise the facilitation of dialogue between different systems' components. Frincu (2011) adopted the use of RabbitMQ as communication mechanism between agent modules to implement a self-healing MAS scheduling platform. RabbitMQ is open source software that implements the Advanced Message Queuing Protocol (AMQP) standard. It has the advantage of not requiring prior knowledge of the number of registered exchanges.

Exchanges are AMQP entities where messages are sent, in other words, a message routing agent. As messages are published to exchanges, any module bound to any exchange by a queue can read messages from it. Moreover, exchanges can take a message and route it to one or more queues. The routing algorithm used depends on the exchange type and rules called bindings (a link between a queue and an exchange). Exchanges are declared with a number of attributes, the most important of which are:

a. Name;

b. Durability (exchanges are safely recovered when failure occurs);

c. Auto-delete (an exchange is deleted when all modules have finished using it);

d. Arguments (these are platform-dependent)

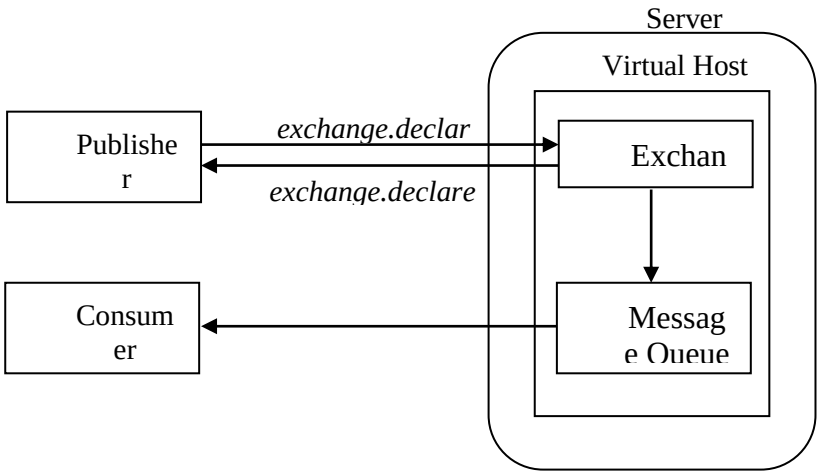


Fig. 4.2: AMQP Model: RabbitMQ Communication Structure

Let us take a look at the exchange class, a group of methods related to operations on exchanges. It includes the following operations:

a. exchange.declare;

b. exchange.declare-ok;

c. exchange.delete;

d. exchange.delete-ok

The operations above form logical pairs: exchange.declare and exchange.declare-ok, exchange.delete and exchange.delete-ok. These operations are "requests" (sent by clients) and "responses" (sent by brokers in response to the aforementioned "requests").

The proposed object-oriented scheduling system employs the use of MAS to schedule parallel jobs in a distributed resource environment. The MAS relies on RabbitMQ for message routing from one agent layer to another, or between two or more agents

designated to perform specific scheduling roles within a specific module. Using RabbitMQ for message routing provides some advantages that are essential to our system. It can be integrated with our object-base storage system that already contains lists of object definitions and characteristics. Moreover, RabbitMQ has a built-in feature that allows for agent publishing and checking the availability of every existing module.

The process involved in the message routing procedure only requires the binding of all agent modules to an exchange and the implementation of the corresponding listener. The scheduling system messages are only routed to a designated sub-set of agents, with targeted scheduling performance role depending on agent layer. For example, job scheduling messages are first routed to the appropriate scheduler module, by using a routing key specific to each scheduler IP address. The essence of this is to enforce a rule whereby agent notification events are only valid within agent parent module. However, there are instances when a message needs to be directed to agents outside their parent module, for example, routing of messages between inter-provider agents modules.

4.1.3 Agents Interaction

Messages sent between module components are encoded using JavaScript Object Notation (JSON) and it consists of five different component parts which include (Frincu, 2011):

a. sender ID (routing key that initiates a message exchange, a universally unique identifier)
b. recipient ID (routing key initiated by the consumer to receive the message)
c. processing name (represents the operation a job is required to perform)
d. message content (consists of actual job input data)

e. message type (identifies message types, it could represent the value of a message type expected to be processed in any of the platform module)

Fig. 4.3 represents the general message format for MAS interactions using the object-oriented programming notation. A unique key is required for both parties that send and receive a message. For implementation in Java, a class that represents an immutable universally unique identifier (UUID) is chosen. A UUID represents a 128-bit value.

```
<script language="javascript" type="text/javascript">
<!--
// Message Object Literals

msgObject = {
        senderID : UUID,
        recipientID : UUID,
        processingName :value|null,
        msgContent : [value1,value2,...,valueN],
        msgType : [value1, value2, ..., valueN],
        method : function(){alert("Method had been called" + this.msgType)}
};

msgObject.method();
alert(msgObject.msgType[2]) // will yield the content of value3

//-->
</script>
```

Fig. 4.3: Sample MAS Message Format

For any job scheduling system, usually the following type of message should be expected, for example, the type of tasks that is given to the platform, "New_Task" or "Reschedled_Task". What this implies is that, a new task can be submitted into the platform for processing by any of the platform modules and at the same time an unscheduled task can be resubmitted for rescheduling. Other types of message structures can be represented in the following form:

a. msgType = "Completed_Task": An execution status indicating that a task has been successfully scheduled and executed.

b. msgType = "Aborted_Task ": An execution status indicating that a task has been aborted and should be presented for rescheduling where necessary.

c. msgType = "Resschedule_Task": This is a message type informing the scheduler that a task needs to be reprocessed.

d. msgType = "Agent_Registration": message specifying a request for a new agent registration.

e. msgType = "Agent_Activation": message specifying request to activate an existing agent.

f. msgType = "Agent_Delete": message specifying a request for agent deletion or termination.

g. msgType = "Module_Registration": specify request for a new module registration

h. msgType = "Module_Activation": specify request for existing module activation

i. msgType = "Module_Destroy": specify request for module destruction.

j. msgType = "Execution_Status": specifies module execution progress and states

While processing instructions provided in the form of message contents and processing parameters might differ, depending on platform and communication mechanisms chosen, here, a similar approach is adopted. Fig 4.4 depicts a code snapshot of processing parameters containing job information, supplied to job agent belonging to a class of scheduling module.

```
<script language="javascript" type="text/javascript">
<!--
// Scheduling Processing Literals

msgObject = {
        [...]
        processingName :{
                                        "senderAgentID": UUID,
                                        "recipientAgentID": UUID,
                                        "workloadDescription":value,
                                        "workloadDependencies":
                                        [value1,value2,...,valueN],
                                        "workloadPriority": value|-1,
                                        "estimatedExecutionTime":value|-1,
                                        "executionStatus":[value1,value2,...,valueN],
                                        [...]
        method : function(){alert("Method had been called" + this.processingName)}
};

msgObject.method();

//-->
</script>
```

Fig. 4.4: Message Content Containing Job Information

Similarly, system or platform monitoring agent which is also part of the scheduling system agent, assumes full responsibility of gathering information that describes the dynamic or static nature of the system environment (ranging from information covering cluster, machine and jobs states). It retrieves this information from the distributed resource platform environment and then forwards it to an appropriate scheduling agent for the next scheduling process execution phase. Shown in Fig 4.5 and Fig 4.6 are both information format and information sample that describes message content representation, as conveyed by monitoring agent to scheduling agents.

```
<script language="javascript" type="text/javascript">
<!--
// Platform/resource information

msgObject = {
        [...]
        msgContent :{
                                "number_of_clusters": value,
        "cluster_id ": value,
                                "cluster_name": value,
                                "number_of_machines": value,
                                "number_of_CPUs_on_one_machine": value,
        "CPU_speed": value,
        "MaxQueues": value,
                                [...]
        method : function(){alert("Method had been called" + this.msgContent)}
};

msgObject.method();

//-->
</script>
```

Fig. 4.5: General format for a message containing machine information

```
<script language="javascript" type="text/javascript">
<!--
// Platform/resource information

msgObject = {
        [...]
        msgContent :{
                                "number_of_clusters": 11,
        "cluster_id": [0, 1 ... 10]
                                "cluster_name": [‘bala’, ‘dolphin’...‘zebra’],
                                "number_of_machines": [32, 267... 768],
                                "number_of_CPUs_on_one_machine": [4, 8... 8],
                                "CPU_speed": [22, 24 ... 26],
        "MaxQueues": 2,
                                [...]
        method : function(){alert("Method had been called" + this.msgContent)}
};

msgObject.method();

//-->
</script>
```

Fig. 4.6: Sample Message Content Containing Resource Metadata Information

4.2 The ANN Based Multi-Agent Resource Selection Architecture

The components in traditional Grid framework require human intervention due to lack of intelligence and use of knowledge from historical data. However, with the current

trends and innovations in Grid applications, there is need for an intelligent resource management system, that would automatically make available, set of resources that meet the resource requirement of the Grid user application. Finding required resources that would satisfy application resource requirement, selecting the discovered resources, allocating the resource, and predicting the estimated execution time for the user application on the selected resources also requires some level of intelligence in a typical resource management system environment.

Therefore, to minimize the error resulting from the active involvement of the human expert, there is the need to replace this involvement with an intelligent system. The incorporation of artificial intelligence into the Grid components can establish the required level of intelligence (Fig. 4.7). Component here mean scheduling mechanism or services such as the scheduler, resource management system, scheduling Agents, etc. Such components require the expertise and intelligence in ANN to achieve automatic scheduling of both resources and user application.

Application scheduling and resource management are some of the two primary components that the Grid services carters for. Application scheduling services takes care of defining application workflow tasks and their dependencies. The resource management component however, is responsible for resource provisioning and storing of resource or system configuration information. The problem of human intervention in any of the aforementioned components can be handled by replacing the domain expert with the MAS, which can be trained by using ANN. By applying the intelligence of ANN for instance to the Job Analyser Agent, the essential application information can be automatically extracted and passed on to the Scheduler Agents. Intelligence can also be applied to the scheduler agent that is responsible for the automatic generation of

application workflow schedules, performance prediction of task execution on selected resource, etc.

The application of ANN to resource management Agents is very significant because of the diverse functions of the component. An intelligent resource management system is preferred and required for the analyses of Grid resource configuration. The area of interest is the Grid resource classification. ANN can be used to automatically classify and create an indexed tree of the Grid resources configuration based on the user application resource requirement (Fig. 4.7). Dynamic information on resource configuration is obtained from the Grid Resource Information Service (GRIS) and Grid Index Information Service (GIIS), which are parts of the Globus Monitoring and Discovering System (MDS-2). The classified resource index tree contains a hierarchical arrangement of results from the intelligent ANN based resource classifier. Scheduling Agents would constantly use the indexed tree storage object to find resources that satisfies the user application resource requirement.

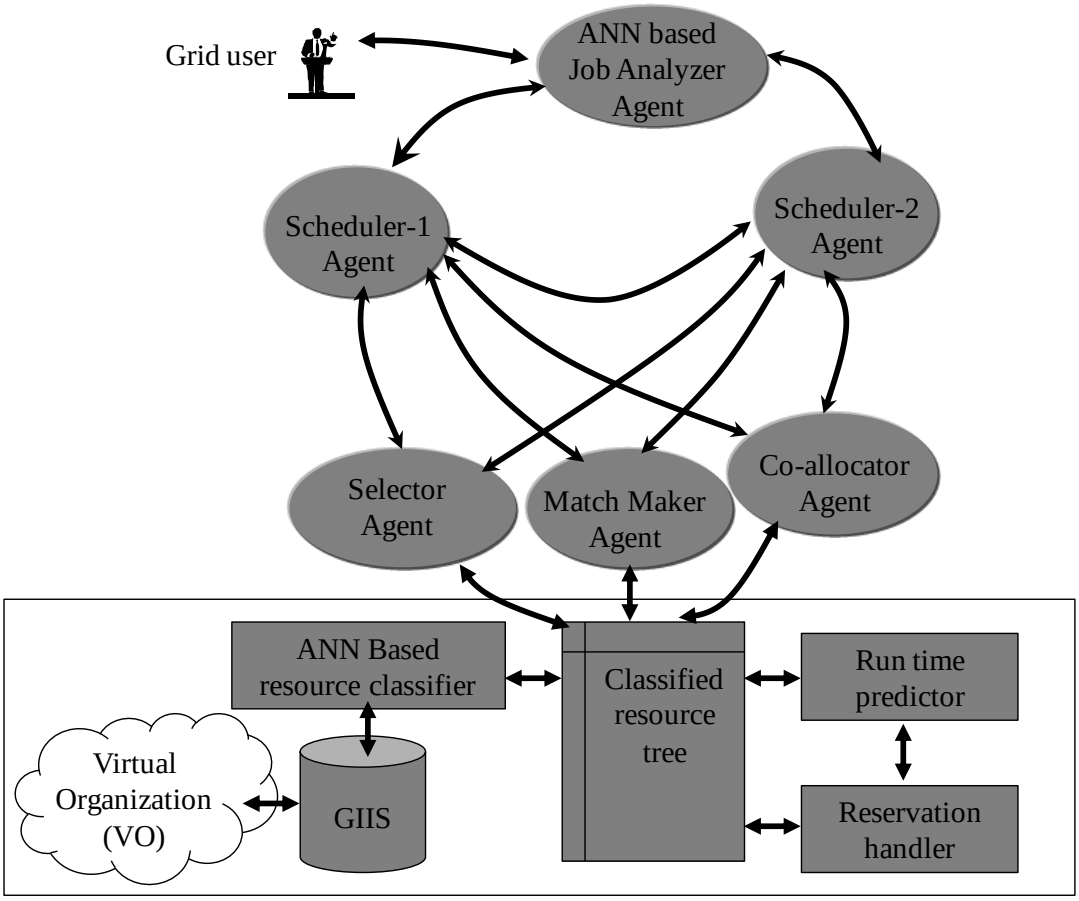


Fig. 4.7: ANN Based MAS Resource Selection Architecture

The replacement of some Grid components with MAS based technology introduces some high level of dynamism into the Grid scheduling system. With planning and coordination, Agents are able to achieve scheduling goals successfully. A Modified Partial Global Planning, which is a coordination mechanism, is incorporated into the proposed MAS scheduling system, to handle the aspect of agent's coordination. In next section, the proposed MAS coordination mechanism, that is required to coordinate the resource selection process is presented. Similarly, the message communication pattern for the proposed agent interaction is described.

4.2.1 MAS Planning through Coordination and Negotiation

Coordination is an important research area in large scale distributed systems resource management. In (Bai *et al.,* 2005), for instance, different coordination mechanisms for Grids are evaluated. From lower to higher level, the following mechanisms are proposed: scripting languages, shared-data spaces and middleware agents, although, the latter was identified to be the most flexible (Lynden and Rana, 2002). The scripting method is quite simple and light weight with respect to agent message mobility and portability.

In MAS, agents cooperate to solve application scheduling problem through negotiation (Durfee, 2014; Salamon, 2011) over their use of local resource, information and expertise. The negotiation procedure can be in the form of team agents negotiating to decide which part of the local problem-solving tasks to pursue, while at other times they negotiate over the distribution of workloads. On the other hand, they might engage in negotiation by reason of wanting to share their information.

Most MAS scheduling uses negotiation protocols for resource allocation and selection of desired scheduling policy, to carter for intra-scheduling process between scheduling modules. The selection criterion is based on a best cost provider approach, where every available policy is evaluated in the light of the current system configuration. Usually, resource providers are expected to publish their configuration information and scheduling policies to the various application frameworks.

The negotiation mechanism considered here is the market-based approach (Lin and Solberg, 1992; Baker, 1991), which used the bargaining strategy to determine each agent capability to do certain tasks. Agent in each group share a common information base referred to as the coordination agent. The coordination agent however, makes use

of a planning strategy referred to as the Modified Partial Global Planning (MPGP) to coordinate each agent action. Ideally, each agent would compute its own processing capability and submit it to the coordination agent, who then reads the resulting information to make decision. After individual agents have made their decisions, they submit their solutions to the coordination agent, which in turn regulates any possible conflicts. Generally, all coordination processes are achieved through the use of the MPGP by the coordination agent in each group. The population of agents is structured into functional groups. These groups are made up of a very dynamic environment, in which agents may enter and leave continuously.

4.2.2 Using MPGP to Solve Resource Selection Problem

The role of agent coordination is very important. It helps in initiating both inter-agent and intra-agent cooperation (especially when dealing with inter-provider scheduling services). The concept of Partial Global Planning (PGP) approach, proposed by Durfee and Lesser (1991) is introduced to handle the aspect of coordinating resource selection problem. In essence, PGP is a general structure for representing coordinated activities in terms of goals, actions, interactions and relationships. It uses a partial global plan evaluation function that is parameterized. The mechanism generally assumes that agents' nodes can be initialized with commonly known scheduling information about their roles, interests, and responsibilities. However, just having knowledge about these sets of information regarding other agents a priori, does not guarantee coordination with respect to finding solution to the global problem. This is so because system conditions change and agents need to be trained and retrained to intelligently deal with the dynamic environment. Therefore, this has prompted the need for a training mechanism for which the ANN-based training technique is suggested and incorporated into this work

afterwards. Given below is the MPGP adopted in this dissertation and used for the agent resource selection and allocation process.

A team of MPGP action can be represented as a tuple of the form:

$$MPGP(\tau) = \langle localAgentId,\ metadata(\tau),\ A_i,\ C(\tau),\ E(\tau)\rangle \qquad (4.1a)$$

Where:

$\tau \rightarrow$ Identifies the current job;

$localAgentId \rightarrow$ Unique name or Id that identifies the local agent;

$metadata \rightarrow$ The information that describes the current job;

$A_i \rightarrow$ Set of agent collaborators;

$C \rightarrow$ Indicates the expected competence of agent to solve τ;

$E \rightarrow$ Indicates the evidence that holds for C

The expected competence of an agent node i to process τ can be determined using the following equation:

$$C(A_i \rightarrow \tau) = k_1 \times cs_i + k_2 \times bw_i + k_3 \times ms_i + k_4 \times np_i \qquad (4.1b)$$

where $k_1 = \dfrac{\beta_1}{\sum_{\forall a_i \in A_i} cs_i}, k_2 = \dfrac{\beta_2}{\sum_{\forall a_i \in A_i} bw_i}, k_3 = \dfrac{\beta_3}{\sum_{\forall a_i \in A_i} ms_i}, k_4 = \dfrac{\beta_4}{\sum_{\forall a_i \in A_i} np_i}$

where:

- A is the set of all agent system collaborators a_i.
- cs_i is the CPU clock speed of agent system collaborator a_i.
- bw_i is the associated communication speed (in Mbps) of the agent a_i.
- ms_i is the memory size of agent a_i.
- np_i is the number of processors that an agent a_i has.

- β_1, β_2, β_3 and β_4 are the weights of the first to fourth terms respectively, assigned to resource parameter to denote significance level, the four weight parameters sum up to one. e.g. $\beta_1 = 0.4$, $\beta_2 = 0.3$, $\beta_3 = 0.2$ and $\beta_4 = 0.1$.

In the initial PGP design proposed by Durfee and Lesser (1991), redundancy and reliability parameters in agent's role were considered major issues. However, in the work of Decker and Lesser (1992), an improvement over the existing mechanism was proposed to tackle these limitations. The coordination mechanism is modified by introducing a computational model, that an agent can use to determine its next neighbour (neighbour here denotes agents of other nodes) competence by using the above competence expression and also an object-based storage infrastructure, that keeps track of up to date information on agent's current process status (in terms of, agents attributes, methods, and behaviour). Considering the overall structure of our proposed scheduling system, agents' roles are categorized into three levels of operations. 1) Agents (called *job agents*) in the first level receive user jobs, according to priorities and user levels, and place them into prioritized global queues. Also on the same level, is the *job selector agent* that sorts the prioritized global queues according to job resource requirements. This information is extracted from the submitted job metadata. 2) *Scheduling agents* at second level perform the tasks of scheduling of user jobs and preparing these jobs for execution. 3) Agents known as the *executor agents* in the third level strictly perform the role of job execution scheduled by the agents in the second level.

Following this structure, agents are not only allowed to exchange and share local information between agents in the same neighbourhood (neighbourhood here implies the level that an agent belong to, as shown in Fig. 4.8), but also are expected to have a partial knowledge of what agents in other levels are working on, are capable of doing

and the evidence that they are able to do what they claim they can do based on their previous records and processing status. The local information in this case would be an exchange of specific abstract query from user on job execution request, based on which agent has certain capability and is also able to perform certain task between the same categories of agent group.

The movement of jobs from one level or neighbourhood to another is usually initiated by agents originally in possession of jobs. The agents initiate job passes, based on their partial view (or knowledge expression) of what the agent in the next level of processing can do. Otherwise, an agent can negotiate for the job it needs from a specific agent at a particular time. This approach can also help reduce reliability and redundancy among active agents.

Fig. 4.8, shows the modified partial global goal plan and coordination relationship structure for the three classified levels of agent activities in the proposed scheduling system. Oval circles are the local goals (or the targeted tasks), implemented by a group of agents belonging to the same class of job description and queries represent partial view of the goal that the agent in the next level is expected to attain. The diagram shows how agents initiate communication by passing jobs metadata to be executed, with their respective goal structures to other neighbouring agents. Coordination relationships are also illustrated among the different agents' levels of tasks.

In the first level, user jobs are submitted to a *job agent*, whose work is to collect and queue those jobs into prioritized global queues. Every user job comprises of some specified action plans or goals, which must be met by the scheduling platform in order to make the execution of the jobs possible. For this, he executes a series of agent tasks. The content of the plans consists of information, ranging from job descriptions (or job

metadata), machine description to computing resource status found in the object-based repository. The *object base storage system* is the framework dynamic data bank that contains fine-grained description of scheduling object instances, such as agent status, network weather information, application and machine descriptions, job metadata, user abstract query that describes a job structure, and so on.

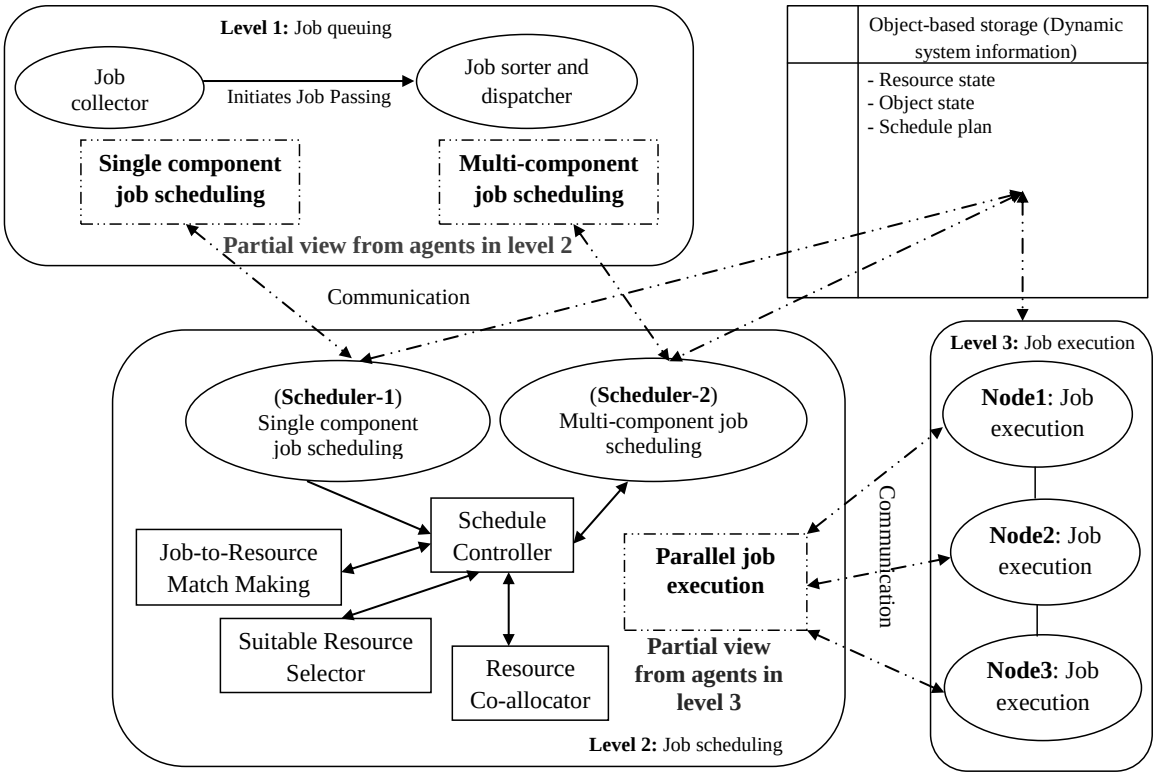


Fig. 4.8: The Agent-Based Coordination Mechanism.

Level 1 consists of two agent tasks, dealing with the initial job processing activities. The first is to collect and queue user job in some prioritized global queues and the second is to sort, classify the jobs according to their job requirement specifications and pass them to an appropriate scheduler in level 2. Level 2 has three agent tasks: the first is to search for suitable resources that will match user job requirements, the second is to sort and match the parallel jobs with the selected distributed resources, and the third task is to generate schedule plans for the execution of the user jobs and then forward the

plan to the selected Grid clusters for execution. Level 3 consists of one major agent task, which is the execution of scheduled jobs.

The modified PGP algorithm invariably serves as a medium in which agents are directed to communicate the top-level scheduling goals. Top-level goal here, mean the overall goal that defines the schedule function that each new local goal is compared with. At every level of the scheduling process, there is usually a local abstract goal targeted to be achieved which is always compared to the global goal. Every solution found for the low-level goals is seen as a local optimum that every agent must attain. This creates an aura of dependence in which certain relationships will trigger further communication.

In Fig. 4.8, the new goal at level 2, which is to schedule selected job, facilitates the top-level goal of level 1, which is level 2's model of activity at level 1. It is therefore desired that the goal at level 2 is communicated to level 1. The partial view of level 2 activity in level 1 is to inform level 1 that tasks of this kind will appear in due course, and by doing so, level 1 thus, will have a partial knowledge of level 2's goal by adding it to its schedule of tasks. Having a partial prior knowledge of agent's capability at the lower-level (level 2), calls for agents at the top-level hierarchy (level 1) to always update the agents in the next neighbourhood, by sending the result of their goals at the time specified in the global goals. The top-level goal update however, is not automatic and might require some additional work which might incur communication overhead on the system performance.

By relying on their ability to learn from previous sequence of scheduling instances, agents can acquire both prior and predicted post scheduling knowledge. Agents have the ability to learn at each stage, the scheduling steps and procedures involved, which

would finally lead to achieving the global scheduling targets. Therefore, it assumed that the scheduling agents can be taught by applying the concept of ANN. The function of the object based storage infrastructure is to supply the ANN with the required historical input data needed for the training functions. An ANN based approach was chosen because, ANNs does not require any language or application-specific knowledge, they learn automatically via periodically updated datasets, and they can provide reliable predictions across a wide range of workload scenarios.

4.2.3 Multi-Agent System Learning with the Backpropagation Algorithm

The backpropagation algorithm is an essential training tool that initiates the leaning process by example. The learning process starts when you introduce into the network an instance of a targeted output goal. The introduction of the sample outputs changes the network's weight in such a way that, at the end of the training session, a desired output is reproduced for every particular input. Fig 4.9a illustrates the backpropagation model.

Generally, input and its corresponding target are referred to as training pair, and are normally represented as (X_q, T_q) where X_q is the input pattern and T_q is the target. The set of inputs supplied to the system effects some changes at each neuron on every layer, until an output is produced at the output layer.

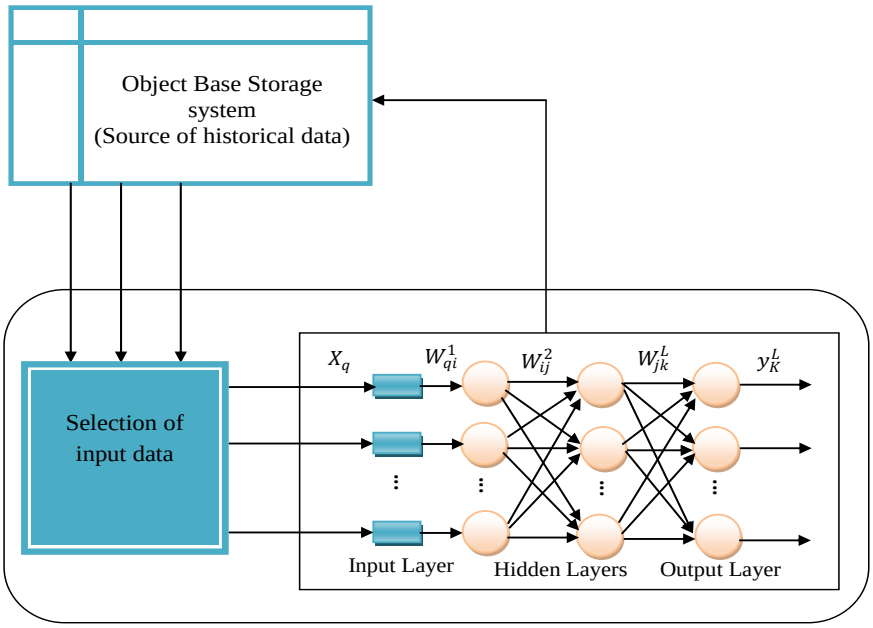


Fig. 4.9a: Backpropagation Network Model

A training set containing user jobs and resource selection examples are used in the learning phase. Figure 4.9a shows a learning process for a group of scheduling agents engaged in the three stages of the scheduling activities explained previously. The three inputs considered in this case are job description, computing resource description, and computing resource status. Inputs here can be viewed as a vector, such as one containing job description given by$< j_1, j_2, ..., j_k >$, computing resource description by $< r_1, r_2, ..., r_n >$, and computing status given by $< s_1, s_2, ..., s_n >$.

There are two basic stages in the learning process when using ANN, the first stage is the learning phase requiring sample input parameters and a dummy target, which represents the desired expected outputs. The source of the input data is the system storage archive (object base repository), that is being updated frequently by agents as the system changes. The second stage is the execution phase, that is performed repetitively until the difference between the dummy target and the computed output is invariably

insignificant. This learning rule is modelled on the basis of the backpropagation algorithm outlined as follows:

a. initialize the weights to small random values between -1and +1
b. randomly select an input pattern from the historic data archive and use it as the training vector input
c. propagate the signal forward through the network and obtain the first output, hence compare the network output with the target output
d. calculate the errors for the hidden layer neurons by backpropagating them from the output layer. This is done by taking the errors from the output neurons and running them back through the weights to get the hidden layer errors
e. repeat step three, by repeating this method, a network of any number of layers can be trained.

The mathematical model presented here explicitly shows all the calculations for a full sized network with three types of input categories, J (job description), R (resource description) and S(resource status), four hidden layer neurons and three output neurons (selected computing resource) as shown in Fig. 4.9b.

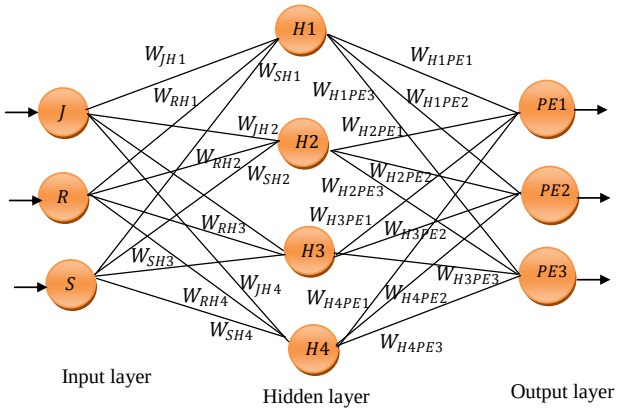


Fig. 4.9b: Detailed Calculation for the Reverse Pass of Backpropagation

a. compute the errors of output neurons

$$\left.\begin{aligned}\delta_{PE1} &= O_{PE1}(1-O_{PE1})(t_{PE1}-O_{PE1})\\ \delta_{PE2} &= O_{PE2}(1-O_{PE2})(t_{PE2}-O_{PE2})\\ \delta_{PE3} &= O_{PE3}(1-O_{PE3})(t_{PE3}-O_{PE3})\end{aligned}\right\} \quad (4.2)$$

Where O_{PEi} is the selected computing resource or processing element 'PE' output at node i, $i = 1,2,\ldots,N \in \mathbb{N}$, t is the target for the selected computing resource or desired output for that node, and δ is the weight error of the selected computing resource output.

b. Change output layer weights

$$\left.\begin{aligned}W^{+}_{H1PE1} &= W_{H1PE1} + \eta\,\delta_{PE1}O_{H1}\\ W^{+}_{H2PE1} &= W_{H2PE1} + \eta\,\delta_{PE1}O_{H2}\\ W^{+}_{H3PE1} &= W_{H3PE1} + \eta\,\delta_{PE1}O_{H3}\\ W^{+}_{H4PE1} &= W_{H4PE1} + \eta\,\delta_{PE1}O_{H4}\end{aligned}\right\} \quad (4.3)$$

$$\left.\begin{aligned}W^{+}_{H1PE2} &= W_{H1PE2} + \eta\,\delta_{PE2}O_{H1}\\ W^{+}_{H2PE2} &= W_{H2PE2} + \eta\,\delta_{PE2}O_{H2}\\ W^{+}_{H3PE2} &= W_{H3PE2} + \eta\,\delta_{PE2}O_{H3}\\ W^{+}_{H4PE2} &= W_{H4PE2} + \eta\,\delta_{PE2}O_{H4}\end{aligned}\right\} \quad (4.4)$$

$$\left.\begin{aligned}W^{+}_{H1PE3} &= W_{H1PE3} + \eta\,\delta_{PE3}O_{H1}\\ W^{+}_{H2PE3} &= W_{H2PE3} + \eta\,\delta_{PE3}O_{H2}\\ W^{+}_{H3PE3} &= W_{H3PE3} + \eta\,\delta_{PE3}O_{H3}\\ W^{+}_{H4PE3} &= W_{H4PE3} + \eta\,\delta_{PE3}O_{H4}\end{aligned}\right\} \quad (4.5)$$

Where W^{+} is the new recalculated network weight, while W is the old weight and the constant η, which represents the learning rate, is usually introduced in order to either speedup or slowdown the learning rate if required. O_{Hi} is the output at each hidden unit i $(i = 1,2,\ldots)$

c. Compute the backpropagation hidden layer errors

$$\left.\begin{aligned}\delta_{H1} &= O_{H1}(1-O_{H1})(\delta_{PE1}W_{H1PE1} + \delta_{PE2}W_{H1PE2} + \delta_{PE3}W_{H1PE3})\\ \delta_{H2} &= O_{H2}(1-O_{H2})(\delta_{PE1}W_{H2PE1} + \delta_{PE2}W_{H2PE2} + \delta_{PE3}W_{H2PE3})\\ \delta_{H3} &= O_{H3}(1-O_{H3})(\delta_{PE1}W_{H3PE1} + \delta_{PE2}W_{H3PE2} + \delta_{PE3}W_{H3PE3})\\ \delta_{H4} &= O_{H4}(1-O_{H4})(\delta_{PE1}W_{H4PE1} + \delta_{PE2}W_{H4PE2} + \delta_{PE3}W_{H4PE3})\end{aligned}\right\} \quad (4.6)$$

d. Change hidden layers weights

$$\left.\begin{aligned} W^{+}_{JH1} &= W_{JH1} + \eta\,\delta_{H1} in_J \\ W^{+}_{JH2} &= W_{JH2} + \eta\,\delta_{H2} in_J \\ W^{+}_{JH3} &= W_{JH3} + \eta\,\delta_{H3} in_J \\ W^{+}_{JH4} &= W_{JH4} + \eta\,\delta_{H4} in_J \end{aligned}\right\} \quad (4.7)$$

$$\left.\begin{aligned} W^{+}_{RH1} &= W_{RH1} + \eta\,\delta_{H1} in_R \\ W^{+}_{RH2} &= W_{RH2} + \eta\,\delta_{H2} in_R \\ W^{+}_{RH3} &= W_{RH3} + \eta\,\delta_{H3} in_R \\ W^{+}_{RH4} &= W_{RH4} + \eta\,\delta_{H4} in_R \end{aligned}\right\} \quad (4.8)$$

$$\left.\begin{aligned} W^{+}_{SH1} &= W_{RH1} + \eta\,\delta_{H1} in_S \\ W^{+}_{SH2} &= W_{RH2} + \eta\,\delta_{H2} in_S \\ W^{+}_{SH3} &= W_{RH3} + \eta\,\delta_{H3} in_S \\ W^{+}_{SH4} &= W_{RH4} + \eta\,\delta_{H4} in_S \end{aligned}\right\} \quad (4.9)$$

4.2.4 Multiple Input Backpropagation Algorithm for ANN Training

The backpropagation algorithm is used to evolve the weights of the feedforward neural network with three layered structures. Assuming that the input layers have three input patterns, the first being job description with k node, the second input pattern is the resource description having m node, and the third input pattern which is the compute resource status with n node. The hidden layers which correspond to the three inputs have q, u, and z hidden nodes. The output layer has p output nodes.

Suppose that the hidden transfer function is sigmoid function (the sigmoid transfer function is given by transfer function = $1/(1 + Exp[-sum])$, (Anderson and McNeil, 1992), then from Fig 4.9b, it can be seen that the output of the k^{th}, m^{th} and n^{th} nodes are:

$$O(A_k) = {}^{1}\Big/\left(1 + e\left(-\left(\sum_{i=1}^{N} W_{ki}.j_i - theta_k\right)\right)\right) \quad (4.10)$$

Where, $k = 1, 2, \ldots, q \in \mathbb{Z}$

$$O(B_m) = {}^{1}\Big/\left(1 + e\left(-(\sum_{i=1}^{N} W_{mi}.r_i - theta_m)\right)\right) \quad (4.11)$$

Where, $m = 1, 2, \ldots, u \in \mathbb{Z}$

$$O(C_n) = {}^{1}\Big/\left(1 + e\left(-(\sum_{i=1}^{N} W_{ni}.s_i - theta_n)\right)\right) \quad (4.12)$$

Where, $n = 1, 2, \ldots, z \in \mathbb{Z}$

N is the number of the input nodes, W_{ki}, W_{mi}, and W_{ni} are the connection weights from the i^{th} node of the inputs layers to the k^{th}, m^{th}, and n^{th} node of the hidden layers, $theta_k$, $theta_m$, and $theta_n$ are the thresholds of the k^{th}, m^{th}, and n^{th} hidden layer unit, $j_i, r_i, and\ s_i$ are the i^{th} inputs; A_k, B_m, and C_n are the weighted sum of inputs for k^{th}, m^{th}, n^{th} units respectively.

$$A_k = \underset{W_k}{\longrightarrow}.\underset{j_k}{\rightarrow}, \Rightarrow \sum_{i=1}^{N} W_{ki}.j_i - theta_k,\ B_m = \underset{W_m}{\longrightarrow}.\underset{r_m}{\rightarrow}, \Rightarrow \sum_{i=1}^{N} W_{mi}.j_i - theta_m,$$

$$C_n = \underset{W_n}{\longrightarrow}.\underset{s_n}{\rightarrow} \Rightarrow \sum_{i=1}^{N} W_{ni}.j_i - theta_n$$

Where $\underset{j_k}{\rightarrow}, \underset{r_m}{\rightarrow}, \underset{s_n}{\rightarrow}$ are the inputs vectors for nodes k, m, and n (j_{ki} = i^{th} input on k^{th} node; r_{mi} = i^{th} input on m^{th} node; and s_{ni} = i^{th} input on n^{th} node), $\underset{W_k}{\longrightarrow}, \underset{W_m}{\longrightarrow}$, and $\underset{W_n}{\longrightarrow}$ are the weights vectors for nodes k, m, and n (W_{ki} = weight on j_{ki}, W_{mi} = weight on r_{mi}, and W_{ni} = weight on s_{ni})

The output of node k output layer is given by $O_k = \delta(A_k) \Rightarrow \sum_{i=1}^{N} W_{ki}.O(A_k) - theta_k$, where $k = 1, 2 \ldots O \in Outputs$

The squared error cost-function is calculated as follows:

$$E = \frac{1}{2}\sum_{p=1}^{N} \left\| PE_i^p - t_i^p \right\|^2 \quad (4.13)$$

Where, N is the number of total training samples, $PE_i^p - t_i^p$ is the error difference in the actual output and targeted output of the i^{th} output for p number of training samples.

The mathematical model presented here, explicitly shows all the calculations for a full sized network with three types of input categories, $J\{j_1, j_2, \ldots, j_k\}$, $R\{r_1, r_2, \ldots, r_m\}$, and $S\{s_1, s_2, \ldots, s_n\}$, n hidden layer neurons and n output neurons as shown in Fig. 4.10a.

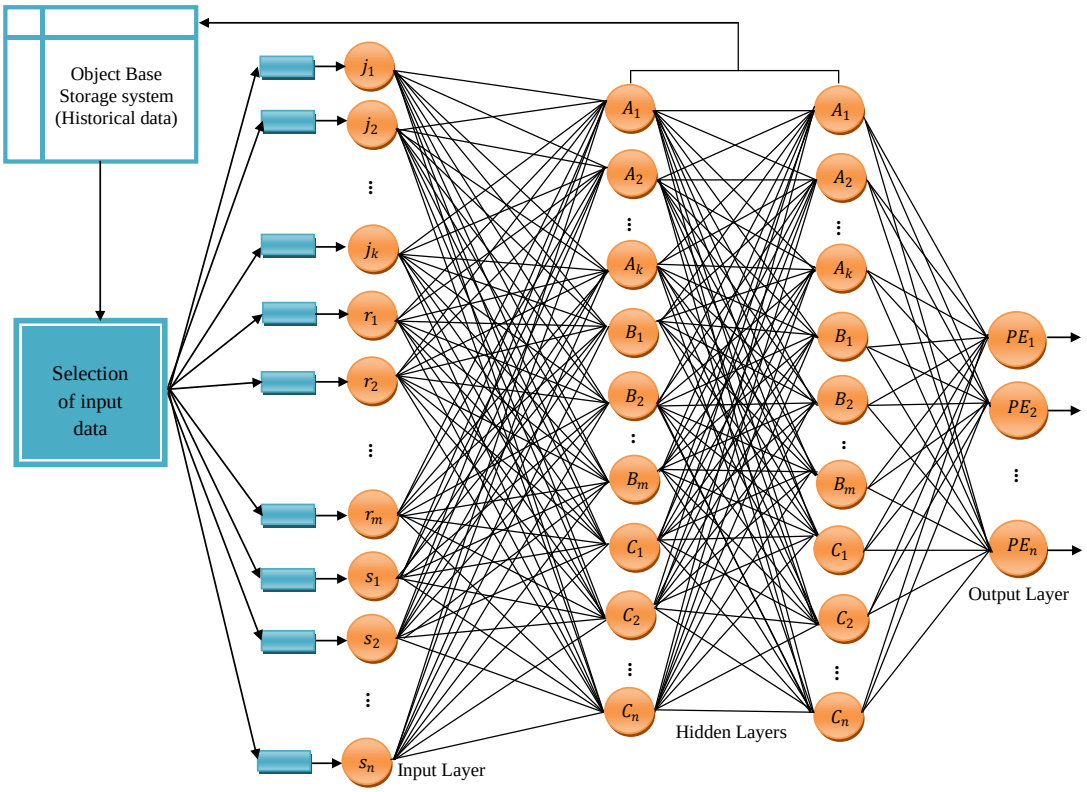


Fig. 4.10a: ANN Based MAS Learning Model

4.2.5 ANN Resource Selection Behaviour

Most often, the task of scheduling process is performed by an expert system user, who chooses the best computational resource according to both job type and framework status. This expert user is in many cases replaced by an automated heuristic which is usually not described in a formal way. Although a formal description is not always available, a study on a suitable set of resource selection examples, can give rise to a

formal generalisation. Since this is possible, agents can be equipped with the set of generalised knowledge by training a process to replace the system expert. In this dissertation, ANN is deployed as a training model for the agent's system.

A study of a user resource selection model based on neural network mechanism has been carried out, and subsequently, simulation of the process was also done. As an example, the user behaviour has been modelled using the algorithm listing 1 described below. It requires the following types of input parameters, $R\{r_1, r_2, \ldots, r_m\}$ and $J\{j_1, j_2, \ldots, j_k\}$. The first set of parameters describes the processing element status vectors. The job description is represented by the second set of parameters. In the simulation, a neural network backpropagation training algorithm is used; this training process is based on supervised learning method. Fig. 4.10b, shows the training behaviour for the ANN resource selection mechanism. The training produces an output, which is used by the scheduling agents, to map and assign user application to the selected resources, for execution.

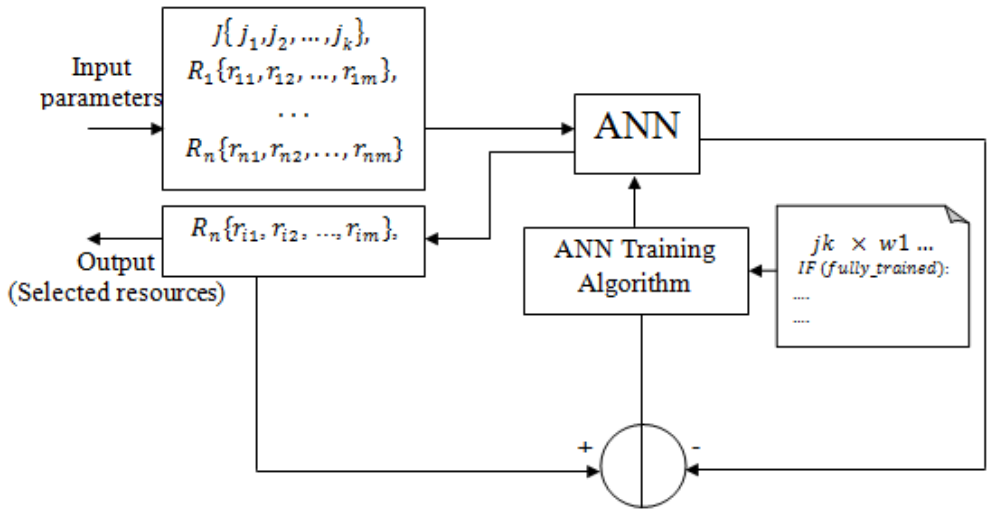


Fig. 4.10b: ANN Resource Selection Behavior

The backpropagation algorithm for the ANN training is presented in the algorithm listing 1.

Algorithm 1: Pseudocode for the backpropagation algorithm.

1: $fully\_trained = FALSE$
2: $DO\ UNTIL\ (fully_{trained})$;
3: $fully\_trained = TRUE$
4: $FOR\ EACH\ training\_vector = < j_1, j_2, \ldots, j_k, r_1, r_2, \ldots, r_m, s_1, s_2, \ldots, s_n, target >::$
5: $// Weights\ compared\ to\ theta$
6: $a = (j_1 \times w_1) + (j_2 \times w_2) + \ldots + (j_k \times w_k) + (r_1 \times w_1) + (r_2 \times w_2) + \ldots + (r_m \times w_m) + (s_1 \times w_1) + (s_2 \times w_2) + \ldots + (s_n \times w_n) - theta$;
7: $y = sigma(a)$
8: $IF\ y != target:$
9: $fully\_trained = FALSE$
10: $FOR\ EACH\ wi:$
11: $MODIFY\_WEIGHT(wi)\ // According\ to\ the\ training\ rule$
12: $IF\ (fully\_trained):$
13: $BREAK$

The proposed algorithm targets the discovery or finding, allocating and reserving of optimal resource for Grid user application execution. The training target is to automatically select the optimal resource set, based on the useful information extracted from the user abstract query submission, through match making process. Resource reservation is done by the ANN based resource classifier component, which stores the classified resource in an indexed tree structure. The advantages of the indexed resource tree are that it provides optimal resources and reduces the time taken by schedulers to find, match and allocate resources to users' application, in addition to resource reservation, for future application scheduling tasks.

4.2.6 Case Study

A case study of the proposed scheduling model is presented next. In our simulation scenario, a set of heterogeneous clusters with the following simulation parameters and configuration values are considered. The simulation was implemented in MATLAB.

Table 4.1: Simulation Parameters

Parameter	Values
Number of clusters per Grid	10 – 100
Number of servers per cluster	4
Number of nodes per server	4-16
Number of processors per node	2-8
Processing power per processor	Intel dual-core CPU @5.0-10.0 GHz
Job Length	100000 - 1000000 MI
Number of jobs	100-9000
Periodic information exchange	20 Seconds
Number of agent groups or team	20% of Node per cluster
Deadline [or max job time slot]	10 – 100 Seconds
Bandwidth	250MB – 1.5 GB
Maximum training length	3500 rounds [epoch]
Learning rate	0.92111
Resource availability probability[0,1]	1

In this simulation, different job samples of parallel MPI applications were tested on various resource combinations per cluster, and compared the measured speedup with the predicted speedup. The simulation experiment was conducted using various job and resource configurations parameters as described in Table 4.1. The sample neural network input layer contains 16 nodes, while the output layer consists of a single node in which the output depicts the best processing element 'PE' selected, according to the simple indicator expressed as follows:

$$O = \frac{\|PE_i\|_2}{n+1} \approx \frac{\sqrt{\sum_{i=1}^{n}|PE_i|^2}}{n+1} \tag{4.15}$$

where, n represent the number of components for the processing element PE status vectors.

For the neural network training and testing phase, two sets, each comprising of 5,000 examples of job submissions have been used. The training phase was terminated at the 3,500 epoch when the network over-training was observed (Fig. 4.11). After the training phase, a validation of our neural network model was performed using a minimum sample of 10,000 to maximum sample of 100,000 job submissions. The training set was presented to the neural network software iteratively, until the network has satisfactorily

learned about the relationship between application resource requirements, and available resource configuration in the distributed system environment. The performance graphs of the trained neural network are shown in Fig. 4.11 and Fig. 4.12, while the regression between the actual and the target output is depicted in Fig. 4.13. Tables 4.2 and 4.3 show the experiments performance evaluation results.

Table 4.2: Experiments of Iterations for the ANN

Learning rate		**0.92111**						
Epochs		500	1000	1500	2000	2500	3000	3500
Evaluation indices	Training MSE	0.1100	0.1050	0.1010	0.0500	0.0110	0.0101	0.0102
	Testing MSE	0.1200	0.1200	0.1200	0.1200	0.1200	0.1201	0.1202

Table 4.3: Performance Evaluation of the ANN Training

Hidden Layer Number	**4**	**6**	**8**	**10**	**12**	**14**	**16**
MSE	0.1100	0.0500	0.0100	0.0100	0.0101	0.0102	0.0102
Time (s)	23.00	23.00	26.00	28.00	32.00	36.00	38.00

Tables 4.2 and 4.3 show that when learning rate = 0.92111 and maximum iterations = 3500, the Mean Square Error (MSE) is minimized for training and testing in the back-propagation ANN. This was performed for six validation checks. When iterations reach the maximum of 3500 epoch, the algorithm terminates the training. The simulation is run 6 times with the same number of hidden layers of 16.

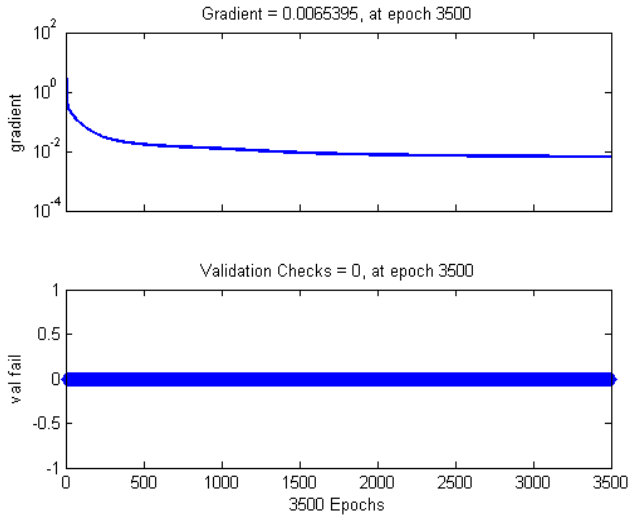


Fig. 4.11: ANN Training State

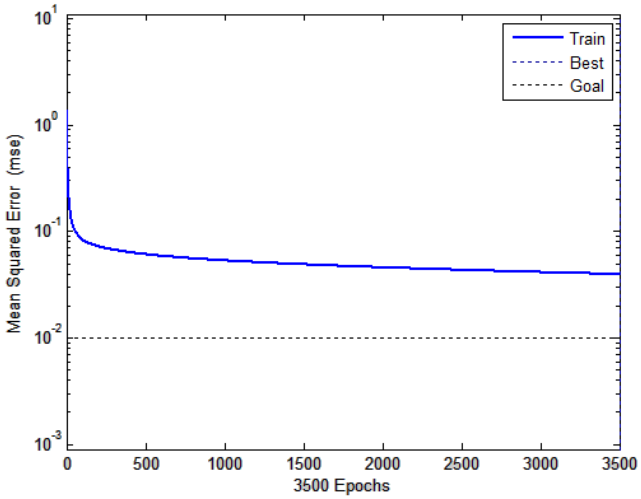


Fig. 4.12: ANN Training Performance Result

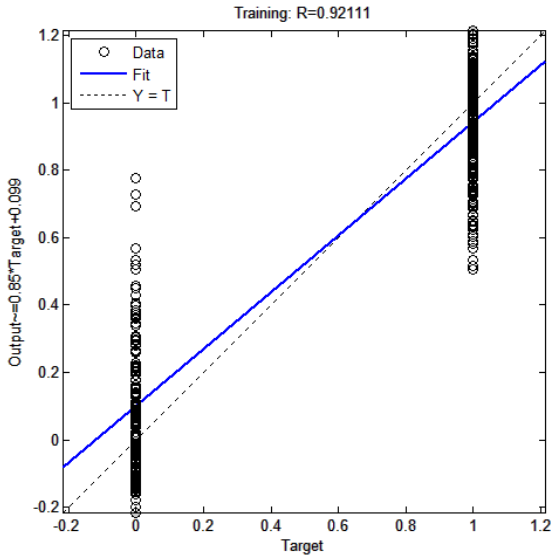


Fig. 4.13: Regression Graph between Actual and Target Output with Training Rate = 0.92111

The performance of the proposed system is determined, by comparing the neural network selection method against the expert user selection as shown in the graphs presented in Fig. 4.14 and Fig. 4.15. The efficiency of the system generally has been evaluated considering the error frequency obtained, as the difference between the norm of the processing element selected by the user and the norm of the processing element selected by the neural network based model. Table 4.4 and Table 4.5 show the performance comparison in seconds between the proposed ANN-based selection method and five other selection methods (Linear Search (LS), Binary Search (BS), Binary Search with Bubble Sort (BSBS), Adaptive Random Search (ARS), and Opportunistic Resources List Indexer Search (ORLIS), for both small and large set of resource list of 500 and 1000 nodes.

Table 4.4: Performance Comparisons for 500 Set of Resources

Search methods	**LS**	**BS**	**BSBS**	**ARS**	**ORLIS**	**ANNS**
Max. time (s)	0.55	.72	0.78	0.68	0.58	0.42

Table 4.5: Performance Comparisons for 1000 Set of Resources

Search methods	**LS**	**BS**	**BSBS**	**ARS**	**ORLIS**	**ANNS**
Max. time (s)	31.44	36.71	29.98	31.72	20.12	17.87

The expert user often selects the first ranked resource on the list of sorted resources. The resources are usually ordered by means of sorting algorithms (example quick sort or bubble sort), which places the resources one on top the other, by swapping according to their ranks of computing ability. The sorting, swapping and selection process is repeated iteratively between adjacent pair of available resources until the best candidate resource is found. However, this system has a serious setback in scalability issue as the number of available resources increases.

The duration of search length based on the Binary Search selection technique is represented by the keyword *BS*, the search length based on the Binary Search selection technique with Bubble Sort is represented with the keyword *BSBS*, the search length based on the Adaptive Random Search selection technique is represented with the keyword *ARS*, the search length based on the Opportunistic Resources List Indexer Search selection technique with bubble sort is represented with the keyword *ORLIS* (Ali and Farrag, 2006), and the search length of the *ANN-b*ased selection technique (which is the proposed method), is indicated by the keyword *ANNS*.

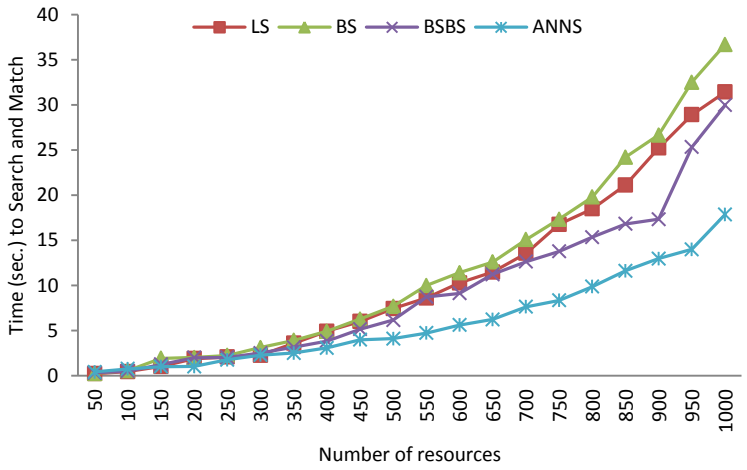


Fig. 4.14: The Search Speed of *BS*, *BSBS* and *ANNS* over *N* Set of Distributed Resources.

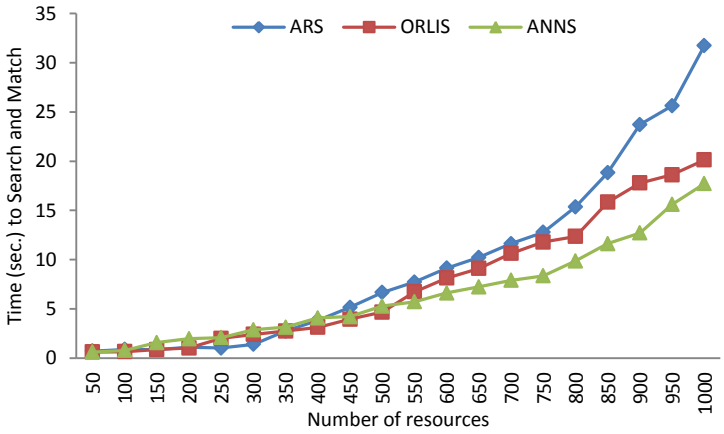


Fig. 4.15: The Search Speed of *ARS*, *ORLIS* and *ANNS* over *N* Set of Distributed Resources.

As it is apparent throughout the simulation results as shown in Fig. 4.14 and Fig. 4.15, the search speed for *ANNS* outperforms all the alternative searching methods. The *ANNS* performs its search by just comparing two adjacent pairs of resources through the set of available resource in the clusters. The search is quickly terminated once a resource that best suits the user request is discovered at the top of the resource list. The advantage of the proposed search method is that, it can result in exceptional performance and scalability, regardless of size, for all frequently requested resource types.

In Fig. 4.14, *BS* search speed appears to have the least speedup, followed by the *BSBS*. At the beginning, the search speed of the two methods appears to be relatively moderate for only small number of resource sets, but it deteriorates as the number of resources scales up. The *BS* search method works by comparing user resource request to the resource at the middle of the available set of resources in the cluster. The comparison determines whether the selected resource is equal to the user resource requirement, less than the user resource requirement or greater than the user resource requirement. When the resource being compared to is equal to the user resource requirement, the search stops and typically returns the position of the resource. If the element is not equal to the user resource requirement, then a comparison is made to determine whether the user resource requirement is less than or greater than the targeted resource. Depending on the nature of the outcome, the algorithm then starts over but only searching the top or bottom subset of the resource list. However, the *BSBS* search performs better than the *BS*, because of the ordering brought by the inclusion of bubble sort. Despite this, our method still outperforms all the alternatives due to the added intelligent and the adaptive nature of the selection process.

ARS is a direct search method that does not require derivatives to navigate the search space. The strategy for adaptive random search is to continually approximate the optimal step size required to reach the global optimum in the search space. This is achieved by trialling and adopting smaller or larger step sizes only if they result in an improvement in the search performance (Masri *et al.*, 1980). While the strategy for ORLIS is to first create an indexer for the available resource list, in form of a table that consists of rank values and number of resources, with rank pairs, and afterward finds a suitable resource, whose rank value is either equal to or greater than the user resource requirement rank values. Result from Fig. 4.15 shows that the search length for the ORLIS is shorter than that of the ARS, the reason being that the strategy used by the ORLIS involves identification of numeric rank values that optimizes the search speed, while ARS uses Stochastic Optimization to determine its search result that is compared to the local optimal.

4.3 Inter-Site Scheduling Technique

In chapter three, the concept of routing indices (RI) was introduced as part of the job distribution mechanism, among peer agents in different virtual organisations or sites. This mechanism is specifically used by the multi-components scheduler agents, to forward scheduling queries from one busy or unqualified peer site to another, in a bid to discover a suitable site, with the resource capability to execute user job. RI query forwarding mechanism is introduced, specifically for inter-site application scheduling and peer agents' interaction.

Routing Indices were initially developed for document discovery in peer-to-peer systems, and have also been used to implement grid information services in (Puppin *et al.*, 2005). The intent of RI basically is to assist system users discover documents with

content of interest across potential peer-to-peer source efficiently (Caminero *et al.*, 2011).

Initiating communication (interactions) among agent peers or nodes in a grid system spanning multiple sites, the RI can be used to make query forwarding decisions between sites in the system, and to avert the need for flooding the entire network. The RI represents the availability of data of a specific type in the neighbour's information base. In Crespo and Garcia-Molina (2002), the hop-count routing index (HRI) scheme was used. This RI scheme takes into account the number of hops (job forwarding) required to find the next documents, and in our case, the best capable peer agents with the right configuration characteristics. In Caminero *et al.* (2011), related version of HRI that is of interest to us was used to determine the aggregate quality of a neighbouring site, based on the number of machines, their power, current load, etc.

4.3.1 Application of Routing Indices in Multi-Agent Scheduling

In this section, the application of routing indices in MAS scheduling of multi-component applications is considered. Depicted in Fig. 4.16 is the architecture of an agent-based multi-component scheduling system, which incorporates a number of components and features described as follows:

a. A heterogeneous resource environment that incorporates agent daemons for each execution site. The agent daemon runs continuously and exists for the purpose of handling periodic job query service requests that an execution site expects to receive. The daemon forwards the requests to other agent daemons in other sites as appropriate.

b. A resource database that provides a fine-grained description of the available machines, associated network bandwidths and storage devices, including capacity and performance parameters.

c. A library of machine performance parameters that automatically configures for other subsequent detected machines.

d. Distributed global schedulers that push scheduling logic to intelligent agents at each execution site to perform global scheduling in a collaborative and coordinated manner. The multi-component schedulers perform three basic functions of, resource discovery, resource mapping and resource selection. The multi-component schedulers, process descriptions of multi-component application task sets, and allocate them to the selected resources for execution based on the available fine-grained component descriptions.

e. A database of multi-component application performance models, which is used by the multi-component scheduler to make decision on which resources is more appropriate for the user submitted application.

The overall system architecture described above is as shown in Fig. 4.16.

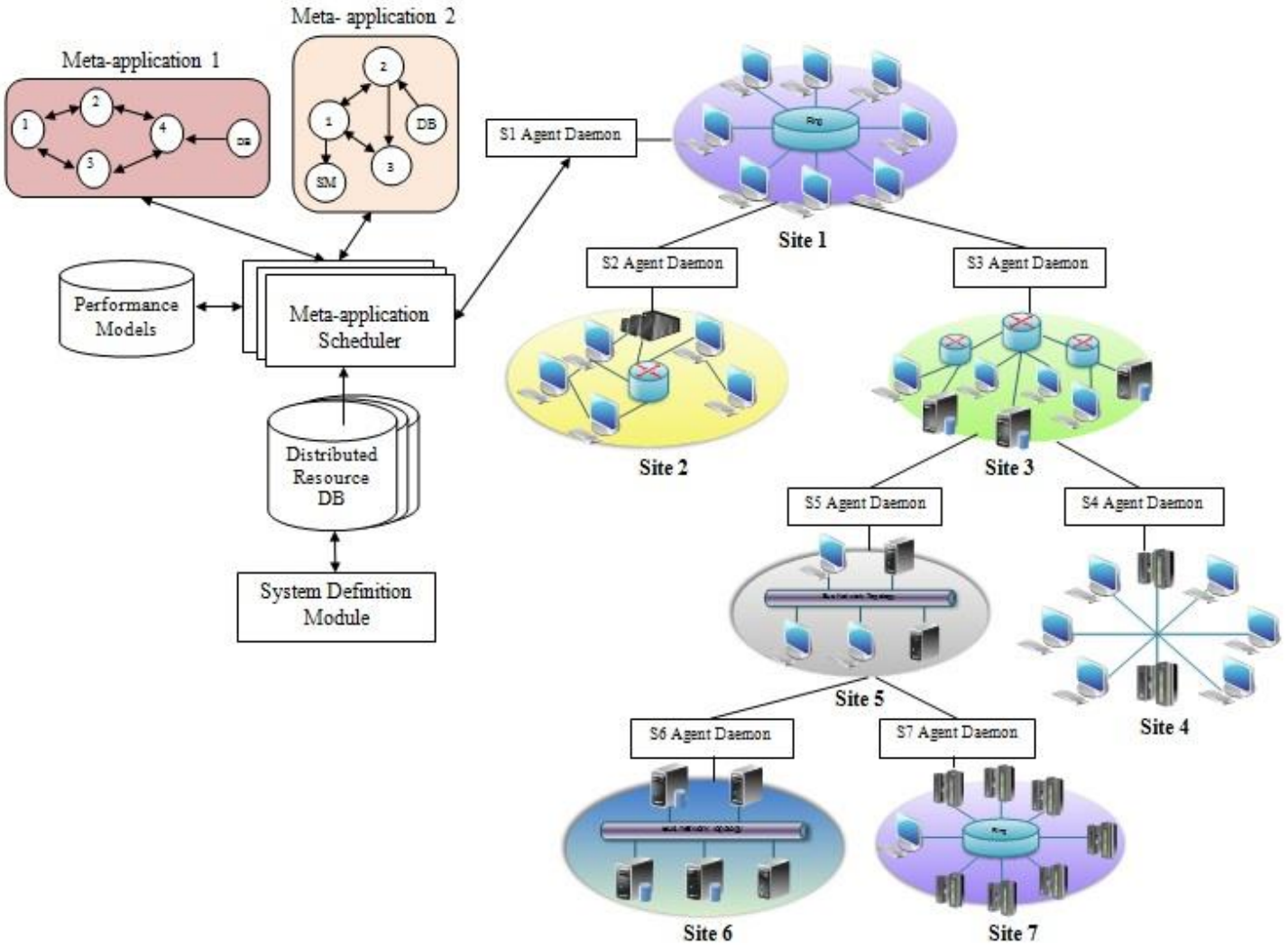


Fig. 4.16: A Multi-Component Scheduling Architecture

At the user level, there is the multi-component applications, that consist of five components in application 2; three computations (1, 2, 3), a database server and a streaming media server (SM), while application 1 consists of five components, in which four are computations (1, 2, 3, 4) and one database server. The presence of an arc indicates data flow.

To explain the scenario presented in Fig. 4.15, a set of agents with specific capabilities that are attached to each execution site is used, to perform global scheduling of tasks that require the use of a certain heterogeneous resources. Let assume that each agent can compute the capability of the site is attached to, and is able to pass the computed information to its neighbours, based on the user query given to it. MAS, generally, comprises of network of intelligent agents, whose, neighbourhood network can be modelled as a graph $AN = (N, E)$, where N is the set of agents and E is a set of bidirectional edges denoted as non-ordered pairs(A_i, A_j). The immediate neighborhood of an agent A_i includes all the one-hop away agents. What this implies is that, $\forall A_j \exists (A_i, A_j) \in E$. The set of agent's A_i neighbors is denoted by $N(A_i)$.

Given a network of agents $G = (N, E)$, and the set of neighbors $(N(A))$ of agent A in N, the routing index (RI) of A denoted by $RI(A)$ is a collection of $|N(A) \cup \{A\}|$, vector of resources and capabilities availability, each corresponding to a neighbor of A or to A itself. Given an agent A_i in $N(A)$, then the vector in $RI(A)$ that corresponds to A_i is an aggregation of the resources that are available to A via A_i. The key idea is that, given a request, A will forward this request to A_i if the resource capability available via A_i can meet the demand of the request.

Let k be the number of resource variables V for machine $M_j \in S$. The variables that can be considered for any resource include, processing speed, memory size, and intra-node

communication bandwidth. To estimate the capability of each site S, a capability function G is defined for machine $M_j \in S$ as a mapping:

Definition 1: Resources capability is defined as a function $G: V \rightarrow M$ which maps every resource variable $V_i \in V$ on a Machine $M_j \in S$ having attached a queue Q_j. Therefore, the function can be expressed as shown in *Eq. 4.16*.

$$G_j(V_1, V_2, \ldots, V_k) \tag{4.16}$$

Where, $V_i, 1 \leq i \leq k$, is a variable of the function.

The capability of a neighbouring site can thus, be computed by an agent using a cost function given in Equation (4.18). For example, a neighbouring site S_j may be preferable over local site S_l for a job execution request requiring compute intensive machines, if the aggregate resource capability of S_j in this regard is higher than that of S_l, based on the number of available processors, processing speed, memory size, and the associated intra-node communication bandwidth. Then, the aggregate resource capability of the targeted neighbouring site S_j is computed by the host agent A_i as follows:

$$I_{S_j}^{S_l} = \left(\sum_{j=1}^{N_S} G_j \right) \times eff\_bw\left(S_l, S_j\right) \tag{4.17}$$

$$G_j = \left(k_1 \times V_1 + k_2 \times V_2 + k_3 \times V_3 + k_4 \times V_4\right) \times N \tag{4.18}$$

where

$V_1 = ps, V_2 = bw, V_3 = ms, V_4 = np,$ as expressed in *Eq. 4.16*

$$k_1 = \frac{w_1}{\sum_{\forall m_i \in S_j} ps_i}, k_2 = \frac{w_2}{\sum_{\forall m_i \in S_j} bw_i}, k_3 = \frac{w_3}{\sum_{\forall m_i \in S_j} ms_i}, k_4 = \frac{w_4}{\sum_{\forall m_i \in S_j} np_i}$$

where,

- $I_{S_j}^{S_l}$ is the information that the host agent A_i keeps about the neighboring site S_j, which is provided by the neighboring agent $A_j \in S_j$.
- N is the number of machines site S_j has.
- ps is the CPU clock speed.
- bw is the associated machine i/o bandwidth,
- ms is the memory size,
- np is the number of processor per machine,
- $eff\_bw(S_l, S_j)$ is the effective maximum bandwidth (Mbps) between the local site S_l and the neighboring site S_j. The value of this parameter is acquired from the Net Weather Service (NWS) tool in Grid Information Service (GIS).
- w_1, w_2, w_2, and w_4 are four weight constants that show the importance of each normalized parameters respectively, (processor speed, i/o bandwidth, and memory size). The three weight parameters sum up to one. The values of the three parameters are decided by experiments on different combinations of the three machine parameter values. The combination with the best performance is adopted for actual use. The values of the parameter ps, ms and np are acquired by the Meta Directory Service (MDS) tool, while bw is acquired by NWS which is part of Globus GIS tool.

From the expression given in Equation (4.17), $w_1 + w_2 + w_3 + w_4$ represents the maximum number of processing capability a particular machine should have. That is, if a machine with the fastest processing speed, largest memory size and highest i/o communication bandwidth is considered, $w_{1+}w_2 + w_3$ should represent the maximum number of processing capability of that machine. These maximum values of processing

speed, memory size and i/o bandwidth must be propagated between peer machines of neighboring sites, so that all the targeted machines share the same values for them. This is to allow for more objective comparison between resources held by different neighbouring machines.

A local agent in site S_1 with lower aggregate resource quality performs the action of forwarding a job to another agent in higher quality site S_2, by computing the aggregate resource quality of S_2. However, there are other sites within the neighbourhood that are also worth considering before forwarding the job by S_1. For instance, if peer S_1 wants to forward a job to one of its neighbours, it will have to decide between S_2 and S_3, which has a better resource quality and effective network bandwidth. Before RI can be put into use, there is need to first consider one major key component that determines the goodness of path of a neighbouring site, which is the goodness function.

The notion of goodness here simply reflects the quality of the path that leads to a neighbouring site. To calculate the goodness function of a neighbouring site S_i with respect to a job query Q, the expression can be represented as follows.

$$goodness\left(S_i, Q\right) = \sum_{j=1..H} \frac{\varphi_{i[j],Q}}{F^{j-1}} \qquad (4.19)$$

where, S_i is the targeted execution site, $\varphi_{i,[j]}$ is the RI entry for j hop through site S_i, and are j hops away from the local site, H is the horizon for $HRIs$ (that is the maximum number of aggregated RIs for each hop or job forwarding), and F is the fanout of the topology (the maximum number of neighbours a site has).

The value of sites that can be reached through the site φ_i and are j hops away from the local site S_1 (in Fig. 4.17), can be calculated as shown Equation (4.20). A typical

illustration of this is $S_{2.3}$ which represents the quality of sites which can be reached through S_2, whose distance from the local site is 3 hops (shown in table 4.6):

$$\varphi k,j = \begin{cases} I_{S_k}^{S_l}, & when\ j=1 \\ \sum_i I_{S_i}^{S_t}, \forall S_i,\ \ d(S_l,S_i)=j \wedge d(S_l,S_t)=j-1 \wedge d(S_t,S_i)=1, & otherwise \end{cases} \quad (4.20)$$

Where, $d(S_k,S_i)$ is the distance (in number of hops) between S_k and S_i. $\varphi_{k,j}$ is calculated based on the distance from some local site. When the distance is 1, then $\varphi_{k,j}$ = $I_{S_k}^{S_l}$, because the only peer that can be reached from local S_l through S_k within one hop is S_k. Otherwise, for those peers S_i whose distance from the local site is j, the information that each peer S_t (which is the neighbour of S_i) keeps about them has to be added. As such, the HRI of site S_1 will be calculated as shown in table 4.6.

Table 4.6: Detail HRI for a Local Site S_1

Site	S_2	S_3
1 hop	$I_{S_2}^{S_1}$	$I_{S_3}^{S_1}$
2 hop	$I_{S_4}^{S_2}+I_{S_5}^{S_2}$	$I_{S_6}^{S_3}+I_{S_7}^{S_3}$
3 hop	$I_{S_8}^{S_4}+I_{S_9}^{S_4}+I_{S_{10}}^{S_5}+I_{S_{11}}^{S_5}$	$I_{S_{12}}^{S_6}+I_{S_{13}}^{S_6}+I_{S_{14}}^{S_7}+I_{S_{15}}^{S_7}$

Briefly let demonstrate how HRI can be used by an agent in a local site through the topology illustrated in Fig. 4.17.

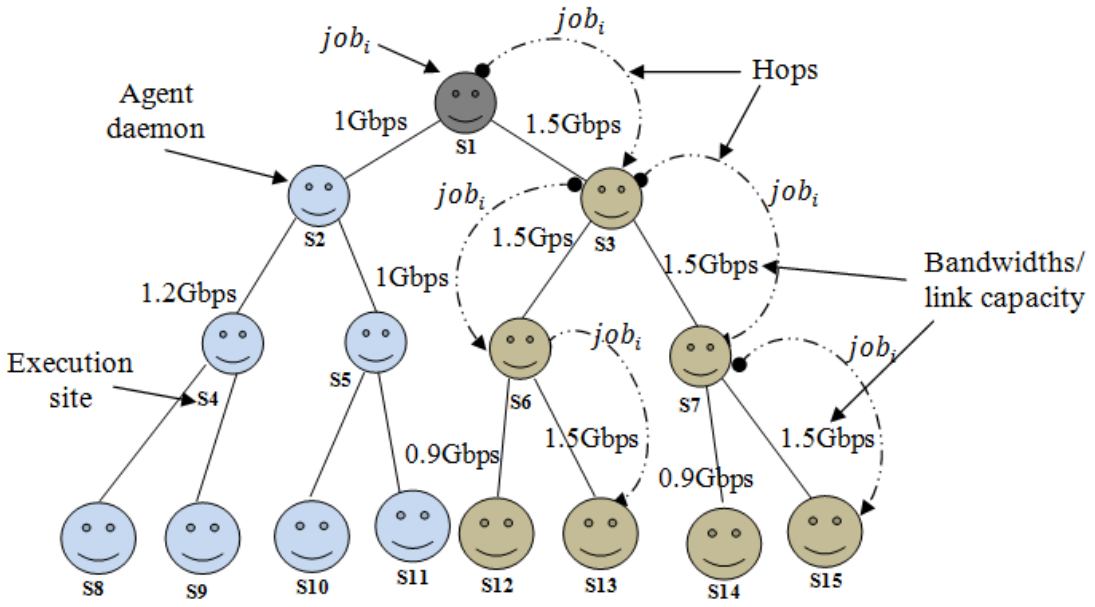


Fig.4.17: Agent-Based *HRI* Inter-Site Relations.

Assuming that all the sites in Fig. 4.17 have 24 nodes, each one with 1 GHz of processor speed, 1 GBps of i/o bandwidth, inter site bandwidths as provided in Fig. 17 and 1 GB of memory. Let the weights $w_1 = 0.5, w_2 = 0.3, w_3 = 0.2$, and the current number of processes be 48 for all of them. The horizon is 3, and fanout is 3. To compute the *HRI* of S_1, each $I_{S_j}^{S_l}$ is calculated as shown in table 4.7.

Table 4.7: Calculation of Routing Indices between each Individual Site $I_{S_j}^{S_l}$

Site	S_2	S_3
1 hop	$I_{S_2}^{S_1} = \left(\frac{12}{48} + \frac{7.2}{48} + \frac{4.8}{48}\right) \times 1 = 0.50$	$I_{S_3}^{S_1} = \left(\frac{12}{48} + \frac{7.2}{48} + \frac{4.8}{48}\right) \times 1.5 = 0.75$
2 hop	$I_{S_4}^{S_2} = \left(\frac{12}{48} + \frac{7.2}{48} + \frac{4.8}{48}\right) \times 1.2 = 0.60$	$I_{S_6}^{S_3} = \left(\frac{12}{48} + \frac{7.2}{48} + \frac{4.8}{48}\right) \times 1.5 = 0.75$
	$I_{S_5}^{S_2} = \left(\frac{12}{48} + \frac{7.2}{48} + \frac{4.8}{48}\right) \times 1 = 0.50$	$I_{S_7}^{S_3} = \left(\frac{12}{48} + \frac{7.2}{48} + \frac{4.8}{48}\right) \times 1.5 = 0.75$
3 hop	$I_{S_8}^{S_4} = \left(\frac{12}{48} + \frac{7.2}{48} + \frac{4.8}{48}\right) \times 0.9 = 0.45$	$I_{S_{12}}^{S_6} = \left(\frac{12}{48} + \frac{7.2}{48} + \frac{4.8}{48}\right) \times 0.9 = 0.45$
	$I_{S_9}^{S_4} = \left(\frac{12}{48} + \frac{7.2}{48} + \frac{4.8}{48}\right) \times 1.5 = 0.75$	$I_{S_{13}}^{S_6} = \left(\frac{12}{48} + \frac{7.2}{48} + \frac{4.8}{48}\right) \times 1.5 = 0.75$
	$I_{S_{10}}^{S_5} = \left(\frac{12}{48} + \frac{7.2}{48} + \frac{4.8}{48}\right) \times 0.9 = 0.45$	$I_{S_{14}}^{S_7} = \left(\frac{12}{48} + \frac{7.2}{48} + \frac{4.8}{48}\right) \times 0.9 = 0.45$
	$I_{S_{11}}^{S_5} = \left(\frac{12}{48} + \frac{7.2}{48} + \frac{4.8}{48}\right) \times 1.5 = 0.75$	$I_{S_{15}}^{S_7} = \left(\frac{12}{48} + \frac{7.2}{48} + \frac{4.8}{48}\right) \times 1.5 = 0.75$

By adding the various results of $I_{S_j}^{S_l}$calculated in Table 4.7, *HRI* values for the site S_1can be obtained as shown in Table 4.8.

Table 4.8: Computed HRI Values for Site S_1

Site	S_2	S_3
1 hop	0.50	0.75
2 hop	1.10	1.50
3 hop	2.40	2.40

In Table 4.9, the goodness function is computed so as to determine the quality of each neighbouring site first before forwarding a job. This is done by computing the quality of sites that can be reached through each neighbour and their distance from the local site. This is to say that, the goodness function determines the effectiveness of each site and only afterward a job is forwarded to the best peer.

Assuming that a computing resource is required at site S_1, then the result of the goodness function computed in Table 4.9 can be used to make an appropriate decision as to which of the two sites to forward a job to. The computed result of the goodness function produces a higher value for S_3 than S_1. This might be due to the fact that the network connection to S_3 makes it more suitable to execute jobs than the link to S_2, in which case S_3 is chosen over S_2.

Table 4.9: Goodness Function Table for Site S_2 and S_3

goodness(S_2)	$\mathbf{\frac{0.50}{3^0} + \frac{1.10}{3^1} + \frac{2.40}{3^2} = 1.13}$
goodness(S_3)	$\frac{0.75}{3^0} + \frac{1.50}{3^1} + \frac{2.40}{3^2} = 1.52$

The importance of the goodness function is obviously very clear and defined. For example, consider the topology depicted in Fig. 4.17. If site S_1 needs to forward a job to one of its neighbours S_2 and S_3, it will have to decide by computation first between the two sites, which has the best quality . In this case, S_1 will apply the goodness function to both of them and one of them will be chosen. When applying the goodness function to S_2, the quality of sites that can be reached through it (namely S_4, S_5, S_8, S_9, S_{10}, and

S_{11}) will be considered. In the same way, the quality of S_3 depends on the quality of S_6, S_7, S_{12}, S_{13}, S_{14}, and S_{15}. This is done by means of the HRI, since it keeps information on the sites that can be reached through each neighbour site. Consider that the best resources belong to site S_6, and all the resources belonging to the other sites are overloaded. In this case, S_1 would choose to forward the job to S_3, because although S_3 does not have a suitable resource, it is closer to S_6 than S_2.

4.3.2 Resource Search Technique

In distributed search mechanism, more effective searches can be performed by agents based on distributed indices. In this configuration, each agent attached to a node holds part of the index. The indexes help agents optimize the probability of finding quickly, feedback to the requested information, by keeping track the availability of required resources at each neighbouring site. In this section, a resource searching technique is proposed from the perspective of peer-to-peer computing. In this context, HRI is incorporated into MAS, to form a search technique that selects quality resources for every job forwarded by an agent from one local site that has no suitable local computing resources, to another neighbouring site, which probably have the available computing resources. Hence, a neighbour site is only contacted if no suitable local computing resources are available.

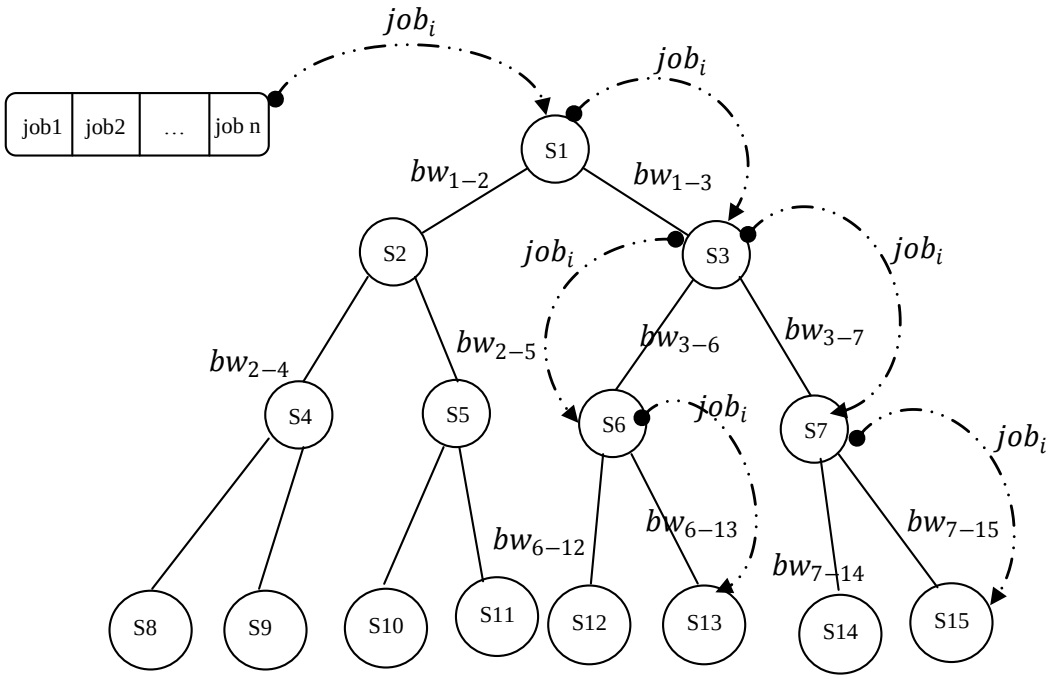


Fig. 4.18: Inter-Site Job Forwarding using RIs Query Forwarding Mechanism.

Fig. 4.18 represents the processes involved when a local agent forwards a job request to an agent in the neighbouring site. Each targeted agent of the neighbouring site calculates its goodness function and also considers the network link quality between the sender site and the recipient sites. As an instance, a job is forwarded from S_1 by the host agent, to the best agent neighbours attached to S_3, S_6, S_7, and S_{15}. These agents compute the capability of each of these sites, until the best site is discovered and job allocation made to the best discovered execution site.

The HRI agent-based resource searching mechanism is formulated purposely for discovering resource with high capability of executing user job. The searching logic is being shifted to agents attached to each of the execution sites, following the tree model presented in Fig.4.18. The distributed scheduling agents make use of this concept in order to first, discover and assign user jobs to the best suited computing resources. It is important to note that in this kind of system, a neighbouring site is only contacted when

the local site does not have enough resources to process the job. In which case, the nearest agent neighbour is contacted for information about its local resources, otherwise, the process continues until a suitable site is discovered and allocated to the requested job.

4.3.3 Resource Search Algorithm

Algorithm 2 is formulated purposely for resource searching and discovery by peer nodes following the architectural model presented in Fig. 4.17 above. The scheduler makes use of this concept in other to perform scheduling of jobs to computing resources. Following these steps, when a job is submitted for execution by a user, usually a fine-grained description of the user job resource requirement is submitted to the scheduler, which searches for a suitable computing resource by sending request to the local site. If a suitable resource is found in the local site, the scheduler automatically generate schedules for that job and assigns it for execution. Otherwise, the job is forwarded to one of the local site neighbour, which is selected based on it goodness function.

It is important to note that in this kind of system, a neighbour site is only contacted when the local site does not have the required computing resources to execute the user job. In which case the nearest neighbour to the local site with the most effective network bandwidth is contacted, otherwise, the process continues until a perfect matching site is discovered and assigned to the requesting job.

Algorithm 2: Resource searching algorithm for multi-component scheduler agents

1: ***Input***: q //new incoming query with job resource requirement description to be processed
2: ***Output***: $NextBestNeighbor$ //a neighbour site selected by the goodness function
3: $LocalResource$ //a resource in the local site with the job description query input
4: $NextNeigbour$: the next neighbor site to forward the query to
5: $\boldsymbol{while}\ q \neq \emptyset\ \boldsymbol{do}$
5: $\boldsymbol{for}\ \forall q_n \in q | i \geq n\ \boldsymbol{do}$
6: $LocalResource := \emptyset$
7: $if\ (QueryStatus(q_n) = \emptyset)\ then$
8: {the first time the query arrives at this site, store the query}
9: $QueryStatus(q) := 1$
10: {look for a computing resource in the local site}
11: LocalResource : = $MatchQueryLocalResource(q_n)$
12: $end\ if$
13: $if\ (LocalResource == \emptyset)\ then$
14: {no computing resource in the local site, so forward the query to a neighbour site}
15: $NextNeigbour := QueryStatus(q_n)$
16: $NextBestNeighbor := HRI(q, NextNeigbour)$
17: $if\ (NextBestNeighbor == \emptyset)\ then$
18: {the query must be bounced back}
19: $Recipient := Sender\ (q_n)$
20: $else$
21: $Recipient := NextBestNeighbor$
22: $QueryStatus(q_n) \mathrel{+}= 1$
23: $end\ if$
24: $ForwardQueryToRecipient(q_n, Recipient)$
25: $else$
26: {tell the requester a computing resource has been found}
27: $SendResponseToRequester(q_n)$
28: $end\ if$
29: $end\ for$
29: $end\ while$

The proposed scheduling framework works with meta-scheduler that comprises of both single-component and multi-component schedulers. Algorithm 2 is designed specifically for the multi-component scheduler using the concept of peer-to-peer network-aware heuristics. The algorithm is used by scheduling agents to perform the task of discovering and searching, for the best execution site, where user's jobs can be executed efficiently.

When a user submits a job for execution, a query with a fine-grained description of the job is first submitted to the multi-component scheduler. The scheduler searches for a

computing resource in the local site that matches the requirements of the query. If the search for suitable resources is successful, then the scheduler generates a schedule that is used to execute the user job. Otherwise, the scheduler forwards the query to one of the neighbour sites. The selection of the next neighbour is based on the HRI. In the case where the forwarded query is bounced back, the second best neighbour is contacted, and the process continues until the next best neighbour is found. Second next neighbour is only contacted in a situation when the current site has no qualified local computing resources. Fig. 4.19 depicts the search process initiated by a local site that lacks suitable local computing resources.

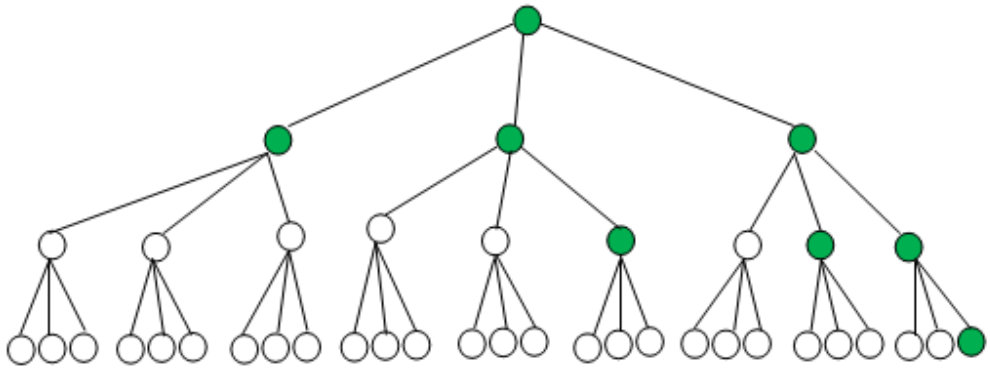


Fig. 4.19: Search Algorithm with a Hop Count of 3

4.3.4 Search Algorithm Comparison

The proposed algorithm scales better than the other two algorithms explained before. In Fig. 4.20, a comparison between the three main algorithms (Distributed Index, Centralised Index and Flooding Search mechanisms) that can be used by the meta-scheduler to schedule a user job for optimal execution is depicted.

The graph shows their position in a diagram that explores 3 different dimensions: job forwarding response time, link network bandwidth and network coverage. The distributed search technique proves to be more efficient than the remaining three

methods in terms of forwarding a query, that contains a job resource requirement, and discovering an appropriate resource peer for the job, it retrieves the result much faster with very low search latency. The flooding technique tends to retrieve the results faster than the single index search, however, it is easy to perceive that it does so at the cost of using the processing power over a large part of the network, whereas for the single index search, although the impact on the network load is significantly lower, the search latency may increase immensely.

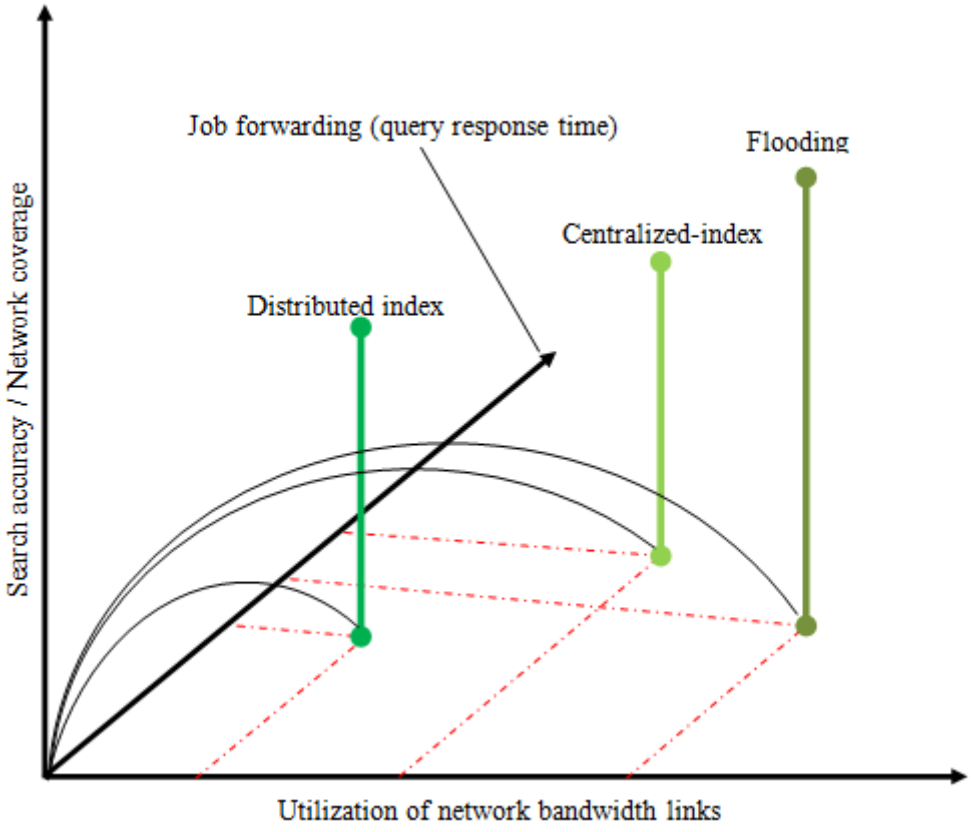


Fig. 4.20: Comparison between Three Resources Searching Techniques, Flooding, Centralised-Index and Distributed Index Search Mechanism.

Time complexity of the algorithm 2

If Fig. 4.19 is assumed to be neighbourhood of site with average branching factor of b times the number of constraints e, then searching a neighbourhood of sites of depth k requires examining

$$1 + b + b^2 + \cdots + b^k = \frac{b^{k-1}-b}{b-1} + 1 = O(eb^k) \qquad (4.21)$$

sites in the worst case. In other words, the search mechanism takes exponential time in the worst case. However, the amount of memory required by this algorithm is linear in the depth of its searches, or $O(k)$, since only currently chosen alternatively at each level of the site needs to be maintained in memory. Hence the number of qualified sites that can be identified by the search algorithm is only limited by the speed of the computer, and not its memory size.

In any case, our algorithm adopts a distributed searching technique, and as such a number of processes per local sites are equally involved in the searching process. A site will naturally consist of a number of these processes. Supposing that a local site has $p = b^k$ number of processes, then each process searches a hierarchy of neighbours to level k. If the depth of the search d is greater than $2k$, then the time required for each process to transverse the first $k\ level$ of the neighbourhood of sites is relatively small, and the search speedup is equally very high.

Assuming that $k = 0$ such that $p = b^k$, then the sequential search can go to level m in the neighbourhood of sites, and each process can explore its own share of neighbourhood originating from sites at level m.

4.4 Job Scheduling and Allocation Mechanism

In today's distributed computing system environments, such as the grid and cloud computing systems, there are instances of having different multi-clusters sites which probably may belong to multiple resource providers each with their own separate schedulers and scheduling policies. In such a case, in order to harness the resource available in these multi-cluster systems together with resident local cluster resource, there is the need to introduce a facility that will be responsible for submitting parallel jobs, that might be requiring these kind of inter provider's resources. Therefore, in our architecture, two meta-schedulers which are responsible for scheduling both single and multi-sites cluster resources are introduced. Jobs are submitted through the local job queue of each individual cluster. The scheduling process starts at the moment when a job arrives at an empty queue.

The scheduling instructions are modularized based on the scheduling activity roles which are triggered and implemented by sets of active intelligent agents' coordination and cooperation, which relies on continuous event execution processes. The set of instructions are described as follows:

a. User's jobs are submitted to the local clusters' job submission queue
b. Job selector agents check whether a submitted job in the queue has characteristics of single component or multi-component and delegate such job to an appropriate scheduler.
c. The job selector agents from onset would usually assign priorities to jobs and cluster, depending on which job enters these queues first, and which cluster is more likely to offer more suitable resources.

d. The scheduler agents, comprising of the single and multi-component schedulers, schedule jobs based on priority policies, by considering both the priority of job selected from a particular cluster, with respect to the cluster's resource contribution priority.

e. The scheduling processes start up immediately when the job is submitted to the cluster local queue.

The interaction diagram can be used to illustrate the way objects in the form of agents in the scheduling system interact with one another. The above scheduling instruction set carried out by group agents, harmonizes the execution process that produces the expected scheduling output. Fig 4.21 depicts agent's interaction sequence diagram.

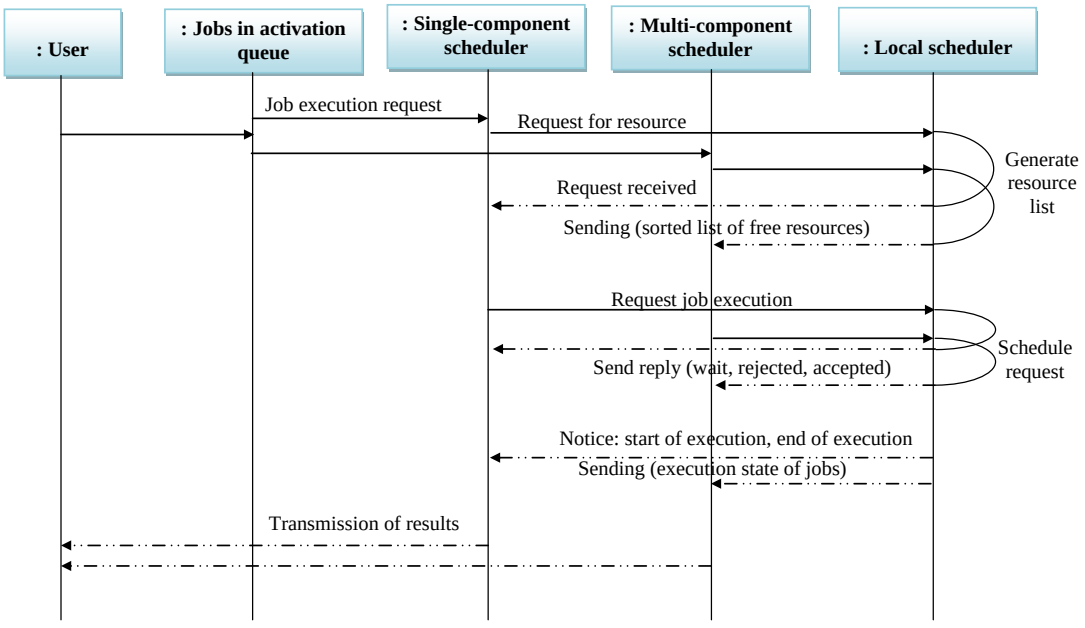


Fig. 4.21: Sequence Diagram Showing Interaction Among Agents Involved in the Scheduling of User Jobs.

4.4.1 Job Agent Module

The proposed scheduling system uses agent-based job distribution strategy at a global level in such a way as to achieve optimum job distribution. Job agent receives the jobs from Grid users, and distributes them among different prioritized global queues, based on user level (Fig. 4.22). The number of global queues can be customized in the grid system according to a certain defined priority classifications. Allocation of jobs to the global queue is strictly based on the priorities assigned to jobs and user level.

One important application of the prioritized global queue is sorting of user jobs (Goodrich and Tamassia, 2006), for example, given a collection of user job J consisting of n number of jobs that can be compared, according to a total order relation. Similarly, the jobs can be rearranged either in increasing order or in nondecreasing order if there are ties. In this section, the algorithm for sorting J with a priority queue Q called $PriorityQueueSort$ is presented. The algorithm consists of the following two steps:

a. In the first step, insert the batches jobs of J into an initial empty priority queue Q, by means of a series of n insert operations, one for each job.
b. In the second step, extract the jobs from Q in nondecreasing order, by means of a series of $nremoveMin$ operations, inserting them back into J in order.

Algorithm 3: $\boldsymbol{PriorityQueueSort(J, Q)}$;

Input: A sequence of J sorting n jobs, on which a total order relation is defined, and a priority queue Q, that compares keys using the same total order relation

Output: the sequence J sorted by the total order relation.

1: ***while*** ! $J.isEmpty()$ ***do***
2: $entry \leftarrow J.removeFirst()$
3: $Q.insert(entry, \emptyset)$ {a null value is used}
4: ***end while***
4: ***while*** ! $Q.isEmpty()$ ***do***
5: $entry \leftarrow Q.removeMin().getkey()$
6: $J.addLast(e)$ { the smallest key in Q is added to the end of J}
7: ***end while***

The implementation of a priority queue Q is simple, let us consider storing some batches of job entries of Q in a list called J, where J can be implemented with a doubly linked list. Thus, the elements of J are entries (k, j), where k is the key (or the index of a job) and j is the value of the stored job.

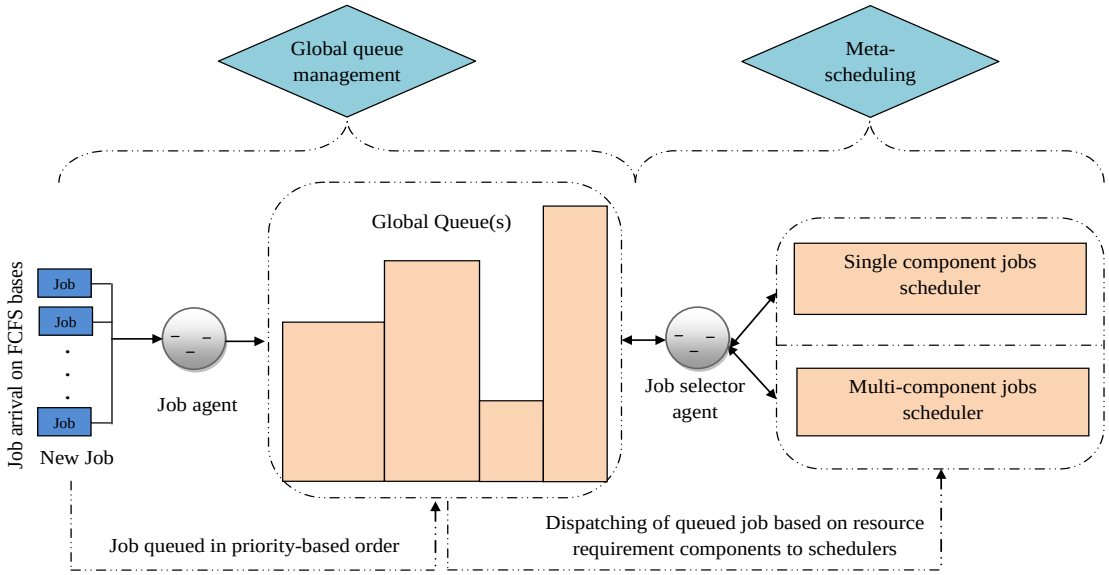


Fig. 4.22: Agent-Based Jobs Dispatching Strategies

The job agent picks up jobs from the grid user based on the concept of First-Come-First-Served allocation policy (FCFS). The prioritized job queuing mechanism is presented in the pseudo-code shown in algorithm listing 4. The algorithm is described as follows: the job agent upon receiving a new job from the user, assigns the job into a set of prioritized global queues $Q_1 \in global\ queue1$, $Q_2 \in global\ queue2$, etc. The global queues priorities are defined based on job priorities and user levels as follows: usually, priority (Q_1) > priority (Q_2), if and only if the priorities of set of jobs in Q_1 is greater than the set of jobs in Q_2. A priority threshold is defined also for each global queue based on a specific job classification, for example, $threshold1$, is defined for global $queue1$ and $threshold2$ is defined for global $queue2$. A job is only assigned to a particular global queue if the priority of the job is $\leq$ to the specified queue threshold.

Algorithm 4: Pseudocode for Priority-Based Agent job distribution

1: $Require: Q_{global} > 1$ //Q denotes job queue
2: $J_{new} = \emptyset$; // a set of new jobs
3: $J_{submitted} \neq \emptyset$;// a set of submitted jobs
4: set $Q_{global}(i) = 0, i = 1,2, \dots, m$
5: $priority\ (Q_{global}(i = 1)) > priority\ (Q_{global}(i = 2)) > priority\ (Q_{global}(i = 3)) > \dots$
6: $k = Threshold\_QueueID$;
7: $jp \leftarrow 0$; // Jp is an auxiliary variables indicating job priority
8: $qp \leftarrow 0$; // qp is an auxiliary variables indicating queue priority
9: **while** $J_{new} \neq \emptyset$ **do**
10: $j_{priority} \leftarrow priority\ (J_{new})$; // the priority of current job
11: $j_{index} \leftarrow index\ (J_{new}) | index: 1,2, \dots, n$
12: **if** $j_{priority} >= jp$ **then**
13: $jp \leftarrow j_{priority}$;
14: $qp \leftarrow j_{index}$;
15: **end if**
16: **end while**
17: **if** $jp < k$ **then**
18: **for** $\forall j_n \in Q | i \geq n$ **do**
19: $submitJobToGlobalQueue(qp); \{Q_{global}(qp) \leftarrow \{j_0, j_1, \dots, j_{k-1}\} | k \in \mathbb{N}\}$
20: **else**
21: $submitJobToGlobalQueue(qp); \{Q_{global}(qp) \leftarrow \{j_k, j_{k+1}, \dots, j_m\} | m \in \mathbb{N}\}$
22: **end for**

The parallel job selector agent performs two tasks, sorting and dispatching of jobs. Jobs in the global queue of the cluster are sorted based on user job resource requirement, and assigned to the appropriate scheduler using two sorting mechanism, Biggest Job First (*BJF*) and Least Flexible Job First (*LFJF*). The job selector agent analyses jobs in the global queue and sorts them, by classifying them into single component jobs and multi-component jobs, based on their resource requirement. Priority here is in accordance with job operational significance. The selection criterion is such that, jobs with large *CPU* demands and requiring more resource than can be provided by a single cluster, are assigned higher priorities and sorted in the order of biggest job first and least flexible job first. These categories of jobs are dispatched to the multi-component scheduler. Jobs with meagre resource requirement that can be satisfied by a single cluster are dispatched to the single component scheduler.

In the global queue, assuming parallel jobs J are submitted in queues by the job agent in the order of $j_1, j_2, \ldots, j_n$, where $j_n \in J$. If a queue that comprises of job list is defined as $Q = \{j_1, j_2, \ldots, j_n\} | n \in \mathbb{N}$, then the job selector agent can then apply this sorting mechanism on the global queue, after which it dispatches the jobs to the appropriate meta-scheduler for the next stage in scheduling and execution of job process. This sorting and dispatching of jobs in the global queue by the job selector agent, is described further in the algorithm listing 5.

Algorithm 5: Pseudocode for assigning a new job to schedulers

$Q \leftarrow \{j_1, j_2, \ldots, j_n\} | n \in \mathbb{N}$
priorityQueueSort (J, Q) /* $J[1:n]$ is an unordered list of jobs to be sorted
according to a criterion $(eg. BJF \ or \ LFJF)$ */
$key =$ ***get_sortindex***();
$n = key$ // *n is the new index of* j_n
job_selector (j_n) // *job selector dispatches jobs to metascheduler*
$scheduler =$ ***reply*** (***job_selector*** (j_n));
if $scheduler \ != \emptyset$ ***then***
for $\forall j_n \in Q | i \geq n$ ***do***
schedule(j_n, Q);
if $scheduler = \emptyset$ ***then***
if $i \ != n$ ***then***
update (Q, i, n); // *update the index of* $j_n (n \leftarrow i)$
job_selector (j_n)
end if
end if
end for
else
return $(failure)$

4.4.2 Formalization of Cluster Resource Mapping

In this section, attention is given to the aspect of distributed resource mapping that associates the resource requirement of the user's jobs, with the availability of cluster machines resource(s). The allocation of grid resource to user job can be modelled, by presenting a formal representation of the process as follows:

Let CL denote a grid resource pool that includes number of processor (*NP*), processor speed (*PS*), memory size (*MS*), and network bandwidth (*BW*), that have to be allocated to some grid user jobs in an integrated form. Let denote the total number of processors in the resource pool by n and the total number of grid jobs (J) by m, then

$$CL = \{NP, PS, MS, BW\}$$

$$NP = \{p_i | \, i = 1,2, \dots, n\}$$

$$Job = \{job_j | \, j = 1,2, \dots, m\}$$

Supposing also that the total number of processors, processor speed, memory size and associated communication speed (bandwidth) are denoted by np, ps, ms, and bw respectively. Then the sum of processors, processor speed, memory size and bandwidth allocated to each job cannot exceed the total available resources.

That is

$$\sum_{j=1}^{m} np_j \leq np \, , \sum_{j=1}^{m} ps_j \leq ps \, , \sum_{j=1}^{m} ms_j \leq ms \, , \sum_{j=1}^{m} bw_j \leq bw$$

However, for each grid job job_j, there is a corresponding minimum requirement of resources that must be met for job_j to be assigned a suitable resource that can execute the job perfectly.

$$\text{Let } job'_j = \{np'_j, ps'_j, ms'_j, bw'_j\} \qquad (4.22)$$

Therefore, the following conditions must hold during resource selection and allocation for any given grid job.

$$np'_j \leq np_j \leq np, j = 1,2, \dots, m$$

$$ps_j' \leq ps_j \leq ps, j = 1,2, \dots, m$$

$$ms_j' \leq ms_j \leq ms, j = 1,2, \dots, m$$

$$bw_j' \leq bw_j \leq \min(bw, bw_j'') \; j = 1,2, \dots, m$$

It is equally important to observe that the assigned bandwidth to a grid job $job_{j,}$ is constrained by both available bandwidths locally bw, and remotely at the job file site bw_j''.

Generalised model for resource matching

In this work, a job $j_i \in J$ is specifically described by its priority based on the amount of resource capacity it will require to be executed on resource $r_j \in R$. Similarly, a resource $r_j \in R$ is described based on its instantaneous available capacity. A resource in this case, can have multiple capacities based on the resource attributes as earlier on described in section 4.4.2. If for instance, the capacity of a resource r_j is described by $\mathrm{H}_{(r_j,t_k)}$ where t_k indicates the type of attribute capacity ($t_k = \{ps, ms, np, bw\}$). And in the same way, we describe the resource requirement of a job j_i by $\Gamma_{(j_i,t_k)}$. A mapping scheme with a set of variables $\mathrm{M}_{(j_i,r_j)}$ and $Z_j = 0, 1$ is described. The variable $\mathrm{M}_{(j_i,r_j)}$ indicates whether a job $j_i \in J$ is matched to the required resource $r_j \in R$ and the variable Z_j, indicates if a job $j_i \in J$ is matched to the required resources.

The following resource capacity constraint can be expressed to ensure that, the sum of the fraction of the resource capacity required by the user applications does not exceed the total available capacity, and to also ensure that the user job is mapped to the appropriate resource for it execution.

$$\sum_{j \in J} \Gamma_{(j_i,t_k)} * \mathrm{M}_{(j_i,r_j)} \leq \mathrm{H}_{(r_j,t_k)} \qquad (4.23)$$

for each $r_j \in R$ and its capacity t_k

The number of resource used to match user jobs can be determined. This is essential were cost is involved, the user may want to limit job execution to a small set of efficient resource to reduce extra cost. Therefore, in order to compute the number of resources used in running user application, a variable S_r is defined with values 0, 1, which is expressed as $S_r = 0$, if resource $r_i \in R$ is not matched to any one of jobs $j_i \in J$, and $S_r = 1$, otherwise, for each $r_i \in R$ by the following two sets of linear inequalities:

$$\sum_{j \in J} \mathrm{M}_{(j_i, r_j)} \geq S_r \qquad (4.24)$$

and

$$\sum_{j \in J} \mathrm{M}_{(j_i, r_j)} \leq S_r N(J) \qquad (4.25)$$

for each resource $r_i \in R$, where $N(J)$ is the number of jobs considered. For a resource r, the sum of $\mathrm{M}_{(j_i, r_j)}$ over jobs is equal to the number of jobs matched to this resources, if no job is matched with this resources, its value is equal to 0, in this case, based on the first inequality, S_r is equal to 0 because S_r must be less than or equal to 0 to make the inequality stand. On the other hand, if some jobs get matched to this resource, the sum of $\mathrm{M}_{(j_i, r_j)}$ over all jobs is greater than or equal to 1. Based on the second inequality, S_r is equal to 1 because S_r must be greater than 0 to satisfy the inequality. Then, the number of used resources can be formalised as a linear expression, $\sum_{r \in R} S_r$

Also, we can define a model for job preferences on resources. There are instances were a job may provide a criterion to rank all satisfying resources. For example, a job may prefer a machine with higher CPU speed and use CPU speed as a ranking criterion, therefore a ranking scheme can be developed, which is expressed as a variable $Q_{(j_i, r_j)}$,

the given rank of resource r by job j . This ranking scheme can be formalised as a linear expression given as follow:

$$\sum_{j\in J, r\in R} Q_{(j_i,r_j)}\, \mathrm{M}_{(j_i,r_j)} \tag{4.26}$$

The matching process utilises this ranking scheme to find a matching scheme containing as many preferred resources as possible.

The basic model for the resource matching problem can be described using the following steps:

1. Let $J = \{J_1, J_2, \dots, J_m\}, i = 1,2, \dots m$ be the set of jobs to be matched to suitable candidate resources.
2. Let $R = \{R_1, R_2, \dots, R_m\}$ be the set of available candidate resources.
3. Every job $J_i \in J$ is assigned a priority P, which depend on the capacity of the required resources attributes $t = \{ps, ms, bw, np\}$.
4. $\mathrm{H}_{(r_j,t_k)}$ defines the capacities of the resource attributes of type t_k for resource $r_j \in R$. Therefore $\mathrm{H}_{(r_j,t_k)} = \sum_{j=1}^{k} t$. Where $t = \{ps, ms, bw, np\} \in \mathbb{R}$.
5. $\Gamma_{(j_i,t_k)}$ defines the equivalent resource capacity required by the user job i for each $J_i \in J$.
6. The variable $S_{i,j} = 0,1$ for each job $J_i \in J$. $S_{i,j} = 1$ if J_i is matched to the required resource $r_j \in R$, and $S_{i,j} = 0$, if otherwise.

4.3.3 Agent-Based First-Fit Resource Assignment Strategy

A Matchmaker agent performs the task of mapping user job components and clusters' resources using the Agent-based First-Fit (ABFF) resource allocation approach, in which case a cluster having the least number of resources greater than or equal to, or

much more than the number of job resource requirement, is chosen and assigned to the user jobs. One advantage of using the ABFF resource allocation strategy is that the ratio of resource availability presented, is at least greater than or equal to the resource request. Invariably, the ABFF algorithm effectively selects the first least partition of resource block that is greater or equal to job component. This approach helps in dealing with the dynamism of distributed resource with respect to machine or node failures, often encountered during job execution. Once machine failure is recorded, the job can be reassigned using the extra available resource where applicable. The agent First-Fit resource allocation strategy is depicted in Fig. 4.23 and the simulation process illustrated in appendix A.

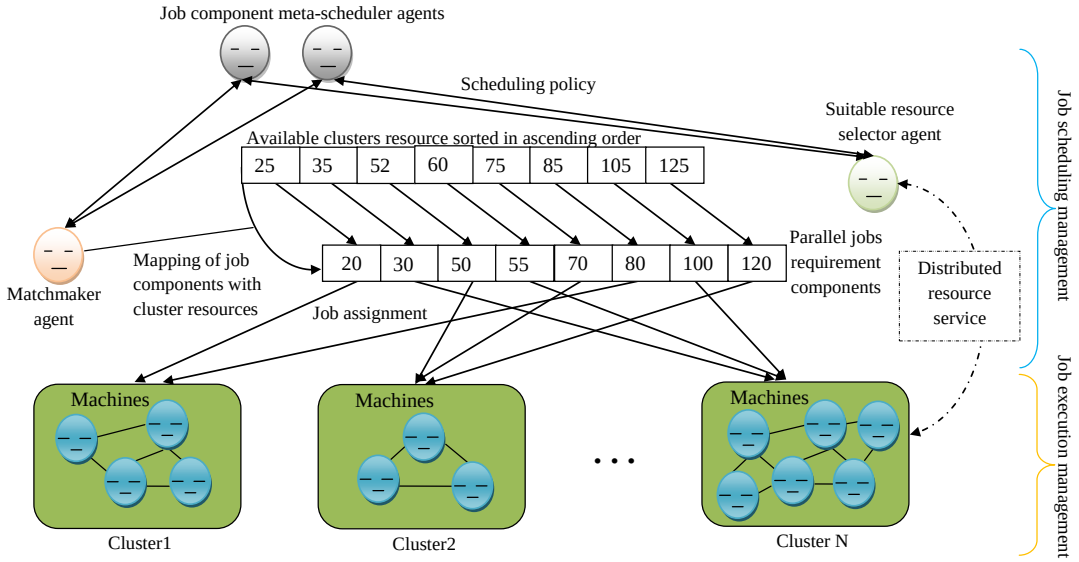


Fig. 4.23: Agent-Based First-Fit Resource Allocation Strategy

In the ABFF allocation strategy, components are sorted in ascending order based on resource requirement and the clusters are sorted in ascending order based on resource availability.

Algorithm 6: Pseudocode for Agent-Based First-Fit resource allocation strategy

1: $\boldsymbol{ABFF}(CL_{ranked}, job_{new})$
2: $CL''_c \leftarrow \{CL'_1, CL'_2, \ldots, CL'_c\}$ set of ranked clusters
3: $\boldsymbol{while}\ CL''_c \neq \emptyset\ \boldsymbol{do}$
4: $set\ CL''(k) = 0, i = 1,2,\ldots,n;$
5: $\boldsymbol{for}\ \forall\ CL'_k \in CL''_c\ \boldsymbol{do}$
6: $job_{index} \leftarrow getIndexOfJob(job_{new});$
7: $CL_{index} \leftarrow getIndexOfCluster(CL'_k);$
8: $job_{selected} \leftarrow job_{index};$
9: $CL_{selected} \leftarrow CL_{index};$
10: $job_{resReq} \leftarrow getSizeOfJobResReq(job_{selected});$
11: $\boldsymbol{if}(\ job_{resReq} \leq CL_{selected}\ )\boldsymbol{then}$
12: $result \leftarrow submitJobToResource(job_{selected}, CL_{selected});$
13: $\boldsymbol{return}\ result;$
14: $\boldsymbol{else}$
15: $job'_{selected} \leftarrow getNewJob(job_{index+1}) | \exists\ job_{resReq} \leq CL_{selected});$
16: $result \leftarrow submitJobToResource(job'_{selected}, CL_{selected});$
17: $\boldsymbol{return}\ result;$
18: $\boldsymbol{end\ if}$
19: $\boldsymbol{end\ for}$
20: $\boldsymbol{if}(\ job_{resReq} > CL_{selected})\boldsymbol{then}$
21: $\boldsymbol{return}\ -1;$
22: $\boldsymbol{end\ if}$

4.5 Resource Ranking Strategy

The ranking of cluster resource is vital, since the rank of a computational node is used as its priority in the scheduling of user job. In a typical heterogeneous computing environment, the execution time of the same user job will differ on different resources, as well as the communication cost via different network links. Invariably this tells us that a computational node assigned to different cluster resources will have different ranking positions. Previous ranking algorithms (Dong and Akl, 2006; Topcuoglu *et al.*, 2002; Sakellariou and Zhao, 2004; Hamid et al., 2006; Radulescu and van Gemund, 1999) did consider some essential factors which had some level of impacts on the quality of scheduling algorithms. However, most of the work concentrated on the ranking of cluster resource based on the heterogeneity of computational resources, their

usage, and the quality of services rendered by the grid resource providers, which most often quantifying these factors, can be a daunting task and more so ambiguous.

However, an improvement can be made by taking into cognizant those factors which highlight the quality of the resource, rather than concentrating only on the usage history of the resource. The factors that project the quality of a grid node emphasize more on the processing capability of the resource nodes, such factors includes the processing speed, memory size, storage capacity and communication bandwidth. Based on these factors, some improvement regarding which suitable candidate resource to retrieve with maximum accuracy can be made.

First, those other factors which are based on ranking of resource are discussed, by considering their usage history and quality of services on the side of resource providers, here, three factors are considered:

4.5.1 Existing Resource Ranking Strategies

The current ranking strategies used by most matchmaking algorithms for the selection of resources include factors which are based on ranking of resource, by considering their usage history and quality of services on the side of resource providers (Brin and Page 1998; Foster *et al.,* 1998; Selvi *et al.,* 2007). In this dissertation, a ranking model is presented, which is based on the justification of resource capability, that strictly relies on some resource performance metrics (such as speed, storage size, number of nodes, memory, and bandwidth). These metrics are as presented below.

4.5.2 Proposed Resource Ranking Strategy

In this section, the proposed resource ranking strategy is presented, which is done through the assessments of grid resource computing capability. More so, we present a

study on resource ranking methods by extracting some set of specific information about the resource specifications and computing qualities. The ranking and selection model is discussed and considered from the perspective of resource performance function.

Resource Performance Function (PF)

In the work presented in (Shih *et al.*, 2009; Shih *et al*., 2007; Yang *et al.*, 2008), the concept of grid node performance ratio was defined in different forms and parameters, according to the requirements of applications. In this work, the proposed algorithm uses the current state of the resource and chooses the resource with the highest performance capability on the basis of the quantitative evaluation. The purpose of calculating the resource performance ratio is to estimate the current processing capability for each potential node. With this metric, the scheduler can select and distribute jobs to the most qualified and freely available grid cluster nodes.

In this study, four resource attributes are adopted. The set of attributes include; processing speed, memory size, number of processor and i/o bandwidth. Then, to rank each available cluster j, according to the order of their respective computation capability, first, the aggregate performance of all the machines $m_i \in j$ is estimated. Therefore, we express the resource performance function pf of a cluster j as follows:

$$pf(j) = \sum_{i=1}^{n} pf(m_i) \tag{4.27}$$

$$pf(m_i) = k_1 \times ps_i + k_2 \times bw_i + k_3 \times ms_i + k_4 \times np_i \tag{4.28}$$

where $k_1 = \frac{\beta_1}{\sum_{\forall m_i \in C_j} ps_i}, k_2 = \frac{\beta_2}{\sum_{\forall m_i \in C_j} bw_i}, k_3 = \frac{\beta_3}{\sum_{\forall m_i \in C_j} ms_i}, k_4 = \frac{\beta_4}{\sum_{\forall m_i \in C_j} np_i}$

where:

- C_j is the set of all candidate machines m_i.
- ps_i is the processor or CPU clock speed of machine m_i.
- bw_i is the communication speed (Mbps) of machine m_i.
- ms_i is the memory size of machine $m_i$and it is a constant attribute.
- np_i is the number of processor in machine m_i, and it is a constant attribute.
- β_1, β_2, β_3 and β_4 are the weights of the first to fourth terms respectively, the four weight parameters must sum up to one. e.g. $\beta_1 = 0.4$, $\beta_2 = 0.3$, $\beta_3 = 0.2$ and $\beta_4 = 0.1$.

The arrangement of the parameters is in order of highest processing priorities. In this configuration, number of processors appears last, since the use of more machines does not result in higher performance, because the high inter-node communication cost outweighs the benefits of greater processing power.

Resource Performance Ratio (PR)

Definition 2: The performance ratio (PR) is defined to be the ratio of all performance functions. Therefore, the cluster with the best aggregate performance percentage ratio is selected. Generally, based on the resource factors under consideration, the rank of each cluster containing a set of machine can be computed as follows:

$$PR(j) = \frac{\sum_{i=1}^{n} pf(m_i)}{n} \tag{4.29}$$

where n denote the number of machines per cluster.

Now, according to this calculated Ranks, user jobs are submitted to the resource(s). The higher the percentage ration of the resource, the higher the priority rate of the resource being selected and assigned a job.

Table 4.10 illustrates, how *PF* is computed, with varying set of resource values. Higher *PF* value for a node means having better performance probability as compared to the other nodes .Thus by applying ranking mechanism, nodes can be ranked according to their PF values, the higher the *PF* value of the resource per node, the higher the node rank. The cluster with the highest collection of rank value nodes is selected for job submission.

Example: Consider the sets of clusters with five set of machines each having a resource attribute specifications as shown below. The performance function is computed using equation 4.27 and illustrated as follows:

Cluster 1: The total respective attributes for all the five machines are given as follows: $\sum np = 18,\ \sum ps = 19353.60,\ \sum ms = 22528,\ \sum bw = 1050$ and the assigned weights as: $\beta_1 = 0.4,\ \beta_2 = 0.3,\ \beta_3 = 0.2,\ \beta_4 = 0.1$

Machine 1 Data: $np = 6,\ ps = 3686.40,\ ms = 4096,\ bw = 250$

$$pf(m_1) = 0.4 \times \frac{3686.40}{19353.60} + 0.3 \times \frac{250}{1050} + 0.2 \times \frac{4096}{22528} + 0.1 \times \frac{6}{18} = 0.5050$$

Machine 2 Data: $np = 4,\ ps = 5324.80,\ ms = 8192,\ bw = 150$

$$pf(m_2) = 0.4 \times \frac{5324.80}{19353.60} + 0.3 \times \frac{150}{1050} + 0.2 \times \frac{8192}{22528} + 0.1 \times \frac{4}{18} = 0.2479$$

Machine 5 Data: $np = 2,\ ps = 5324.80,\ ms = 4096,\ bw = 200$

$$pf(m_5) = 0.4 \times \frac{5324.80}{19353.60} + 0.3 \times \frac{200}{1050} + 0.2 \times \frac{4096}{22528} + 0.1 \times \frac{2}{18} = 0.2147$$

Cluster 2: The total respective attributes for all the five machines are given as follows: $\sum np = 22,\ \sum ps = 16793.60,\ \sum ms = 20480,\ \sum bw = 1050$ and the assigned weights as: $\beta_1 = 0.4,\ \beta_2 = 0.3,\ \beta_3 = 0.2,\ \beta_4 = 0.1$

Machine 1 Data: $np = 6,\ ps = 1024,\ ms = 8192,\ bw = 150$

$$pf(m_1) = 0.4 \times \frac{1024}{16793.60} + 0.3 \times \frac{150}{1050} + 0.2 \times \frac{8192}{20480} + 0.1 \times \frac{6}{22} = 0.1745$$

Machine 2 Data: $np = 4,\ ps = 5324.80,\ ms = 4096,\ bw = 200$

$$pf(m_2) = 0.4 \times \frac{5324.80}{16793.60} + 0.3 \times \frac{200}{1050} + 0.2 \times \frac{4096}{20480} + 0.1 \times \frac{4}{22} = 0.2422$$

Machine 5 Data: $np = 2,\ ps = 5324.80,\ ms = 4096,\ bw = 200$

$$pf(m_5) = 0.4 \times \frac{5324.80}{16793.60} + 0.3 \times \frac{200}{1050} + 0.2 \times \frac{4096}{20480} + 0.1 \times \frac{2}{22} = 0.2331$$

The set of PF values for some randomly generated clusters' data set are also computed as shown in Table 4.10

Table 4.10: Computation of Resource Performance Function and Ranks

Cluster_1

C1Node ID	# processors	Processor speed	Memory size	Bandwidth	Performance function	Rank
node_1	8	5030	9851	109	0.0883	8
node_2	3	6031	7481	108	0.0775	10
node_3	10	5806	15400	180	0.1155	3
node_4	7	5909	15282	137	0.1012	5
node_5	4	5411	6722	176	0.0837	9
node_6	6	5662	6537	200	0.0948	7
node_7	18	5496	4609	159	0.1242	1
node_8	8	5108	13368	189	0.1028	4
node_9	22	3838	3423	113	0.1170	2
node_10	9	3803	11185	200	0.0950	6
TOTAL	**95**	**52094**	**93858**	**1571**		

Cluster_2

C2Node ID	# processors	Processor speed	Memory size	Bandwidth	Performance function	Rank
node_1	22	5030	9851	109	0.1248	2
node_2	3	6031	7481	108	0.0775	10
node_3	10	5806	15400	151	0.1094	3
node_4	7	5909	15282	137	0.0999	5
node_5	4	5411	6722	176	0.0841	7
node_6	2	5662	6537	171	0.0796	9
node_7	30	5496	4609	159	0.1517	1
node_8	8	5108	13368	189	0.1017	4
node_9	14	3838	3423	113	0.0873	6
node_10	9	3803	11185	131	0.0840	8
TOTAL	**109**	**52094**	**93858**	**1444**		

Cluster_3

C3Node ID	# processors	Processor speed	Memory size	Bandwidth	Performance function	Rank
node_1	8	5030	9851	109	0.0883	8
node_2	3	6031	7481	108	0.0775	10
node_3	10	5806	15400	180	0.1155	3
node_4	7	5909	15282	137	0.1012	5
node_5	4	5411	6722	176	0.0837	9
node_6	6	5662	6537	200	0.0948	7
node_7	18	5496	4609	159	0.1242	1
node_8	8	5108	13368	189	0.1028	4
node_9	22	3838	3423	113	0.1170	2
node_10	9	3803	11185	200	0.0950	6
TOTAL	**95**	**52094**	**93858**	**1571**		

From Equation. 4.30, the rank of the cluster j can be computed as follow:

$$Rank(j) = \frac{\sum_{i=1}^{n} pf(m_i)}{n} \times 100 \tag{4.30}$$

where, index $j = 1, 2, 3,$ and $i = 1, 2, \ldots, 10$ which means that, there are three clusters with 10 computational node or machines each. Each of the cluster resource specifications are as listed in table 4.10. The performance ratio of each cluster is calculated using Equation (4.30). The results are as follow: Cluster 3 has the best performance specification with 28% and is ranked one, it is more likely to be selected

among the three sets of clusters. The second cluster has performance ratio of 22% and therefore ranked second, while cluster one is ranked three with 20% performance ratio.

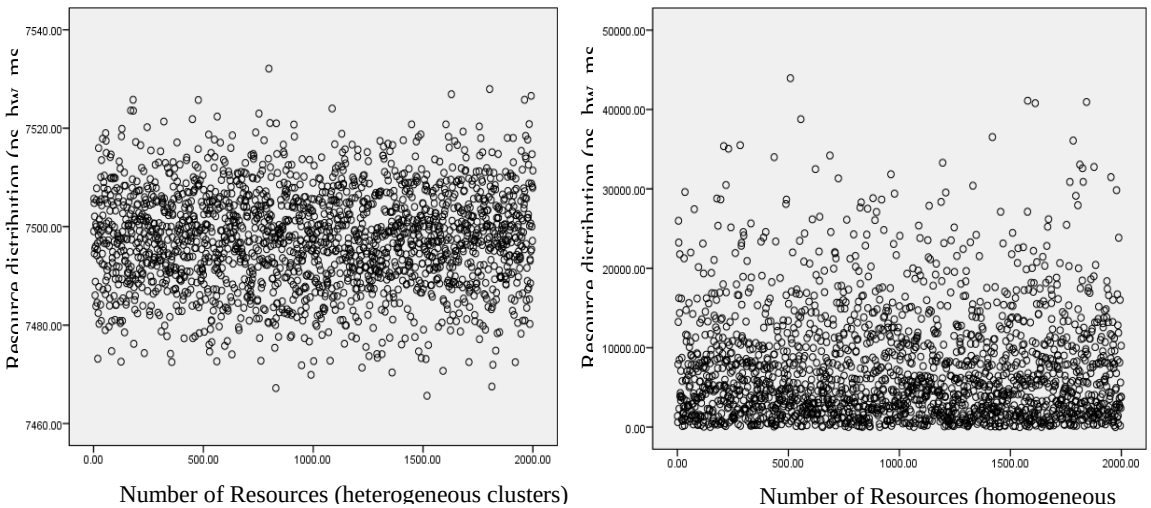
$$Rank(Cluster_1) = 20\%,\ Rank(Cluster_2) = 22\%,\ Rank(Cluster_3) = 28\%$$

Table 4.11: Cluster Global Ranking

Resources	PR (%)	Global Rank	Selection
Cluster_1	20	3	
Cluster_2	22	2	Cluster_3
Cluster_3	28	1	

Algorithm 7: Resource Ranking Algorithm

1: $CandidateResource = \emptyset$;
2: $QualifiedResources = \emptyset$;
3: $ResourcePRofX = 0$; $ResourcePRofY = 0$; $LastRank = -1$; $Rank = 0$;
4: ***while*** $(ResourceQueue\ != \emptyset)\{$
5: ***for*** $\forall X_m, Y_n \in ResourceQueue | i, j \geq m, n$ ***do***
6: $ResourcePRofX = getResourcePRofX(X_i + CandidateResources)$;
7: $ResourcePRofY = getResourcePRofY(Y_j + CandidateResources)$;
8: $Next = getRank(Rank(ResourcePRofX) > rank(ResourcePRofY));\}$
9: $ResourceQueue = ResourceQueue - Next$;
10: $CandidateResource = CandidateResource + Next$;
11: $Rank = rank(CandidateResources)$;
12: ***If*** $(requirements(CandidateResource) == true\ \&\&\ Rank > LastRank)$***then***
13: $QualifiedResource = CandidateResource$;
14: $LastRank = Rank$;
15: }
16: ***if*** $QualifiedResource = \emptyset$ ***then***
17: ***return*** $failure$
18: ***else***
19: ***return*** $QualifiedResource$
20: ***end if***
21: ***end if***
22: ***end for***
23: ***end while***



a. Fine-Grained Resources Distribution b. Coarse-Grained Resources Distribution

Fig. 4.24: Resources Granularity Distribution

In the new system, Fig. 4.24a description of resource is fine-grained, since lower-level details are considered such as CPU speed, i/o bandwidth, memory size and number of processors per node demands of jobs in a 2000-node. In the existing system, Fig.4.24b like condor, description of resources is coarse-grained since only the availability of workstations is considered, and in some other system, resource descriptions end at providing OS type and hardware architectures of clusters. Resource mostly are assumed to be homogeneous for the existing systems even when it is not so.

The placement of a circle indicates the CPU speeds, communication bandwidths and memory resources consumed by tasks. In Fig.4.24a, the resources are normally distributed showing that there is maximum utilization of resource by jobs. While in Fig. 4.24b, resources are partially utilized as a result of non-characterisation of the various resource components. Fig 4.25 illustrates the discrepancies in the amount of times it takes to select a resource for particular job placement. It is faster to sort and select resource when they are characterised and sorted in a particular order of preference.

Normally, resource characterisation helps identify resource suitability based on its computed quality of service. Applications differ in their resource requirements, while some are CPU bound, others might be memory bound or even network bound applications. Hence, having a full knowledge of resource quality, can aid in allocating the appropriate kind of resource to an application.

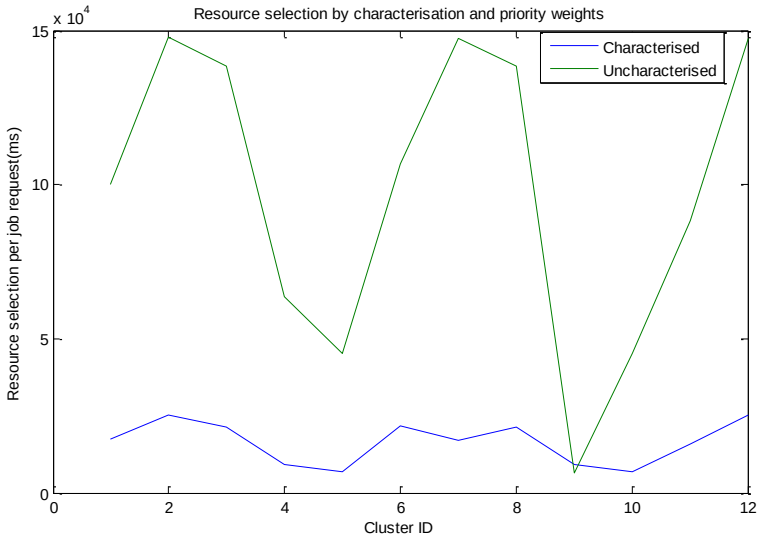


Fig. 4.25: Comparison between Characterised and Uncharacterised Resource Distribution

Typical example scenario from Fig. 4.25 is the case with Cluster 9, with record of highly ranked resource mix, which is uniformly distributed. The low peak is an indication that all its resources have high probability of being selected and thus require little or no characterisation.

A full-fledged general-purpose resource selection framework that strictly relies on resource quality of service for its consumption is developed. The resource selection processes are in two steps. The first step is the initial filtering of resources, with the goal of identifying a list of authorized resources that are available to a given application. These initial lists of authorized resources are further refined by filtering according to the

coarse-grained application requirements, such as hardware configuration (CPU speed, network bandwidth, memory size and number of processors), operating system, and storage capacity.

In the second steps, those resources are aggregated into small collections, such that each collection is expected to provide performance desired by the given application. The number of ways the resource could be aggregated would be extremely large, especially when the number of feasible resource is huge. This can be a limitation to the proposed resource selection method. However, to avoid the complexity, some mathematical models discussed earlier on are introduced, that would help determine the quality of services provided by each resource node. Fig. 4.26 is an illustration of resource performance ratio based on the proposed mathematical model. The performance of a node is determined by computing its aggregate resource configuration, and ranked based on the quality of the computed performance ration.

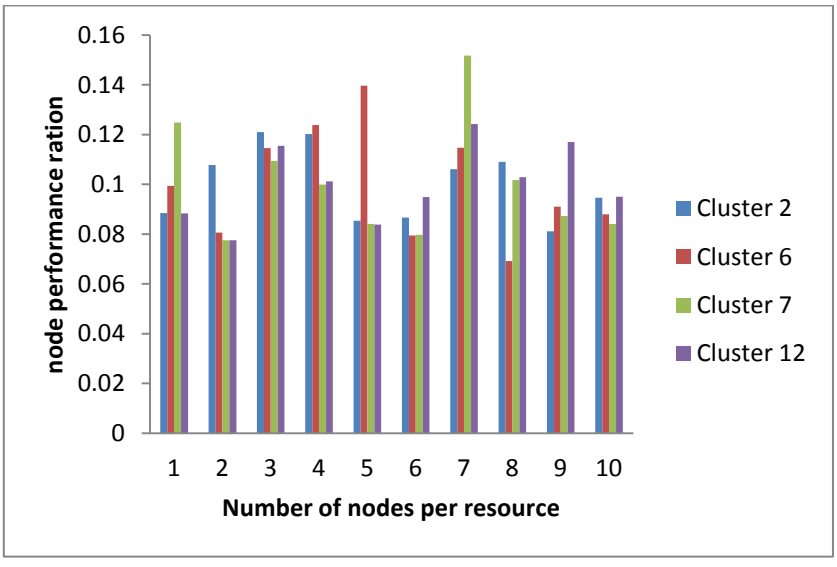


Fig.4.26: Evaluation of Intra-Cluster Resource Performance Quality

In Fig. 4.26, the performance ratios of 10 computing nodes belonging to four different clusters are determined, and assess how their aggregate quality differs. Cluster2 and

cluster12 have more nodes with high performance ration than Cluster6 and cluster7, thus cluster2 would be select as the best qualified followed by cluster12. What determines the quality of a cluster is the consistency in its aggregate nodes PF values.

4.6 Job Resource Requirement Constrained Selection Model

There is no absolute best ranking function that addresses in general, the sorting and selection of suitable clusters resource that best meets the user's jobs. However, implementing a ranking function that evaluates the ranking algorithm for the available cluster resources requires not only considering the cluster's resource attributes, but also putting into consideration the user's job resource requirement characteristics. Ranking and selection operations are the two basic operations that are required for selecting the most appropriate clusters for any user job, first for the ranking operation; it determines where a new resource index fits in the order of some lists of indexed resources, while the selection operation finds the resource index with a given rank. Following the work of (Zhang and Dong, 2000), a selection criteria for group of ranked resources that satisfies the user's job resource requirements is developed.

Suppose the set of all possible ranked functions for a list of available resources is represented by "RANK", and the user job component attributes by "C", then the clusters resource selection satisfaction level (that is the probability of selecting resources that satisfies the user's job components) can be defined as: $C \times RANK \rightarrow [0,1]$, $select(c, rank)$ will be proportional to the user's job resource requirements satisfaction of the rank function. Given a set of cluster resource $CL = \{cl_1, cl_2, \ldots, cl_n\}$, arranged in decreasing order of user's job resource requirement criteria. The 'reverse order number' function of the cluster resource cl_i under the rank function is defined as follow:

$$\psi(cl_i) = |\{cl_k|(cl_k \in CL) \wedge (0 \leq k < i-1) \wedge (rank(Cluster(cl_k)) < rank(Cluster(cl_i)))\}| \quad (4.31)$$

Also given the job components as $C = \{c_1, c_2, \ldots, c_m\}$ and ranking these elements in the same manner as above, gives the expression:

$$\psi(c_j) = \left|\left\{c_q \middle| (c_q \in C) \wedge (0 \leq q < j-1) \wedge \left(rank(Job(c_q)) < rank(Job(c_j))\right)\right\}\right| \quad (4.32)$$

By considering the two operations jointly, suppose that for each j from 0 to $n-1$, there is one exchange and $n-1-j$ compares, so the totals are n exchanges and $(n-1)+(n-2)+\ldots+2+1+0 = n(n-1)/2 \sim n^2/2$ compares.

Then

$$select(c, rank) = \left(\frac{\sum_{i=1}^{n} \psi(cl_i)}{\frac{(n-1)(n-2)}{2}}, \frac{\sum_{i=1}^{n} \psi(c_j)}{\frac{(n-1)(n-2)}{2}}\right) \quad (4.33)$$

4.6.1 Agent-Based Matchmaking Model

The selection of a cluster by scheduling agent is based on node ranking and performance function. Computational node having the best resource characteristics is chosen over those nodes with poor resource processing capabilities. The ranking process is defined by first considering a set of clusters' machines characteristics such as the number of processors, processor speed, memory size, storage size and network bandwidth. First, a model for the agent-based matchmaking resource selection model illustrated in Fig. 4.27 is presented. The system is divided into three processing stages of agent activities which are identified as follows: Job selection and analysis stage, resource selection and match making stage, and resource claiming stage.

Job Selection and Analysis Stage

The first stage requires job agents to select jobs from the global queue and analyse the job for necessary information gathering, which would be required by the scheduler for resource selection, allocation and process execution. This initial information gathering requires a fine-grained description of job structure (such as whether a job is of single component or multi-component and the overall job resource requirements). Fine-grain here means detailed description of job resource requirement. The result of the analysis would then be forwarded to the scheduler for schedule generation report. Schedule generation implies searching, selecting, and mapping any best available resource based on job requirements.

The job requirements consist of a set of dependencies and constraints associated with each dependency. Each requirement is a dependency on a resource instance of a particular resource type. By associating constraints with each dependency, a job may specify the minimum criteria for selecting a resource instance of a particular resource type. For example, a request with a dependency on a computing resource may specify minimum 2.5 GHz processor speed, 500 MB available memory, 4 Mbps of associated network bandwidth as constraints.

Resource Matchmaking Stage

On the side of the grid resource provider, resources are advertised and published via the grid information services (*GIS*). In general, information regarding resource is an important part of the input data to the scheduling performance model, similarly with the job descriptions. The description of resources can be categorized into either fine-grained or coarse-grained. A coarse-grained description provides partial information about resource characteristics, such as resource *OS* type (Linux and Windows) and hardware

architecture (*SISD, SIMD, MISD, and MIMD*) only. The fine-grained resource description lists all detailed information related to that particular resource. For example, a fine-grained resource description may include the resource processing speed, memory size, storage size, i/o communication bandwidth, parallelism type supported, and so on. Generally speaking, a fine-grained resource description is much more useful to scheduler than the coarse-grained.

The resource selector agent is responsible for first sorting, searching and retrieving of the lists of suitable resources based on the order of priority of the resource capability values generated by the resource manager. The match maker agent receives the resource capability report together with the job requirement report, and maps the respective retrieved resources with the jobs and provides an optimal list of potential pairs of resources, and jobs in the form of schedules generation report.

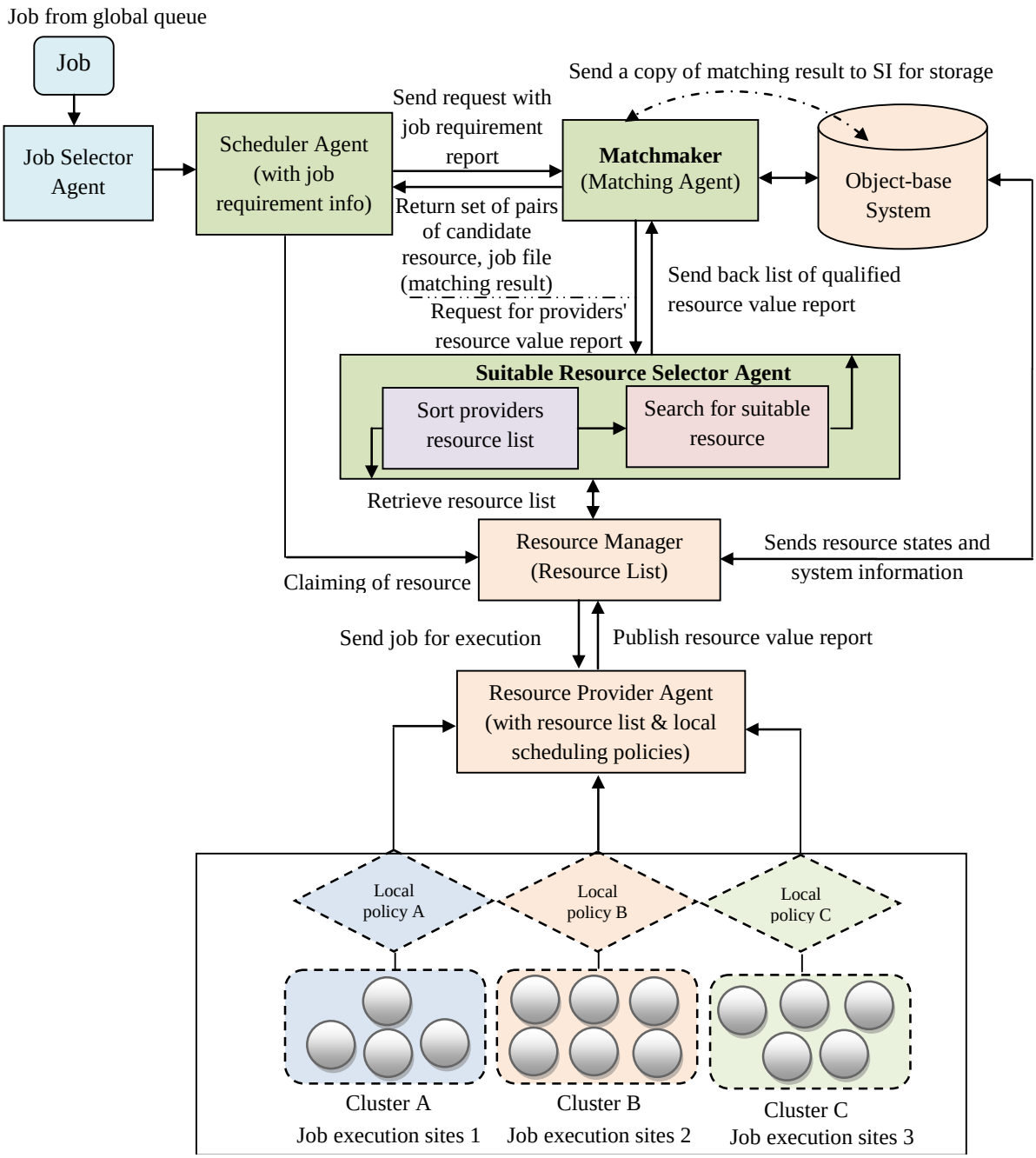


Fig. 4.27: Agent Based Match Making Process Flow Architecture Pattern

Resource Claiming Stage

After a successful pairing of resources with jobs by the Match Maker Agent, the scheduling process enters its final stage of claiming the selected resources at the provider's local execution site. The scheduler sends a pairing report of the matching process that was generated by the match maker agent, to the provider’s resource

manager system. The system then claims the potential execution site and submits the job for execution.

4.6.2 Efficient Resource Sorting Criteria

Resource(s) components in resource lists published by individual resource providers through a resource management system, have to be sorted with special sort criteria to speed up the search for most suitable and available resource process. The aim of the sort criterion is to filter out and keep the most qualified candidate resources, in terms of their resource quality. The filtered candidate resources are ordered according to their computed rank values (based on computing capabilities). Low rank resource(s) may mean low cost resource with minimum attribute specifications. Resource rank values are computed based on the formula given in section 4.5.2. The steps involved in the sort criterion are as follows:

a. The resource having the smallest rank value is sorted to the bottom of the list.
b. If there is equality (or ties) in the ranking value, then resource which has smaller number of unfinished jobs is considered.
c. If the equality continues, then the resource that joined the system first is placed on top the list.

Algorithm 8: Pseudocode for Sorting Advertised Resource List

1: $\boldsymbol{sort}(CL_c)$
2: $\boldsymbol{Input}$: $CL_c = \{m_{1,c}, m_{2,c}, \dots, m_{i,c}\}$ $(where\ i \in 1, 2, \dots, k, and\ c \in 1, 2, \dots, l)$
3: $\boldsymbol{Output}$: $The\ array\ CL_c\ with\ resource\ elements\ rearranged\ in\ non-decreasing\ order\ according\ to\ user\ job\ resource\ requirement$
4: $\boldsymbol{for}\ \forall\ m_{i,c} \in CL_c\ \boldsymbol{do}$ // i denotes machine index and c denotes cluster index
5: $rankValue \leftarrow Rank(m_{i,c})$ // compute the rank of each machine
6: $m'_{i,c} \leftarrow rankValue$
7: $\boldsymbol{for}\ i \leftarrow 1\ to\ n-1\ \boldsymbol{do}$
8: $\{insert\ m'[i,c]\ at\ its\ proper\ location\ in\ m'[0,c], m'[1,c], \dots, m'[i-1,c]\}$
9: $cur \leftarrow m'[i,c]$
10: $k \leftarrow i-1$
11: $\boldsymbol{while}\ k \geq 0\ and\ m'[k] > cur\ \boldsymbol{do}$
12: $m'[k+1,c] \leftarrow m'[k,c]$
13: $k \leftarrow k-1$
14: $m'[k+1,c] \leftarrow cur\{cur\ is\ now\ in\ the\ right\ place\}$
15: $\boldsymbol{end\ while}$
16: $\boldsymbol{end\ for}$
17: $\boldsymbol{end\ for}$
18: $\boldsymbol{return}$

4.6.3 Efficient Resource List Searching Technique

The searching of resource in the object-based storage system which contains large number of entities is an issue, which may slow down the overall resource scheduling process. So, a special searching technique can be used to resolve this problem. Here, a different approach is applied to speed up candidate resource access, which is to store records in sequential order but use a file index mechanism along with the sorted resource list itself. A file index is a list of key/block pairs, arranged with the keys in a specific order.

The resource list entries and the index are arranged sequentially according to the resource rank values. The original records on the resource list can be arranged in any convenient order based on the initial sorting criteria. This usually means that new records are simply appended to the end of the file, so the records are ordered by the time of insertion. The index mechanism however, provides direct access to the sorted

resource list instead of relying on the traditional sequential access method used to find a suitable resource.

An advantage of the indexed approach is that multiple indexes, each with a different key, can be created for the same record file. In one index, the keys can be resource object processing speed, in another, it is resource object storage size, and in another it can be required communication network bandwidth. Because the indexes are small compared to the file, this doesn't increase the total data storage very much. Similar approach was discussed in the work of (Ali and Farrag, 2006).

The searching process is based on the indexing technique that relies on the following steps:

a. Starting with the resource rank value R_1 in the index to the rank value less than the required by one. Next, get the summation of all the resource(s) pair's numbers.
b. The block number gives the summation of all the rank pair's positions which represent the index of the first resource having the required rank value equal to or greater than the job resource requirement rank value.

From the illustration depicted in Fig. 4.28, finding a suitable resource for job of rank 4, the process is done in such a way that the summation = 4 + 2 = 6. This is the total sum of all the jobs ranging from Rank 1 to rank 3 before the required value rank item. As shown in the illustration below, the index 6 is the index of first resource with rank 4 because there is no resource of rank 3. If the example shown in Fig. 4.28 is used to find a suitable resource for job of Rank 6, the summation = 4 + 2 + 3+6 = 15. As shown the index 15 is the index of first resource with rank 5.

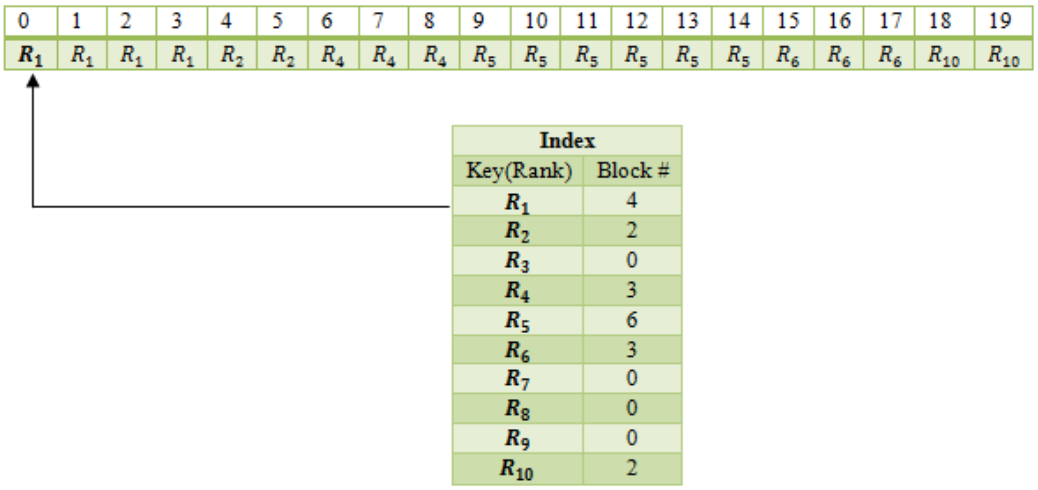


Fig. 4.28: Sorted Resource List and Indexing Mechanism

To insert a new ranked resource item in an indexed file two steps are necessary.

a. First step is to insert its full record into the main file

b. Second step is to insert an entry, consisting of the key and the block number where the new record is stored into the index.

Algorithm 9: Pseudocode for Suitable Resource Searching from the Resource Providers' List

1: ***Algorithm*** $Search(resourceList, jobResReq)$
2: $resourceList = \{m_{1,j}, m_{2,j}, \dots, m_{i,j}\}$ $(where\ i \in 1, 2, \dots, k, and\ j \in 1, 2, \dots, l)$
3: $sortedResourceList \leftarrow sort(resourceList)$
4: $while\ sortedResourceList\ \neq \emptyset\ do$
5: $index \leftarrow 0$;
6: ***while*** $index < sortedResourceList.Count\ and\ sortedResourceList[index]\ \neq jobResReq$
7: $index\ \leftarrow index\ +\ 1$;
8: ***end while***
9: $if\ index\ \geq\ sortedResourceList.Count\ orsortedResourceList[index]\ \neq jobResReq$;
10: ***return*** -1;
11: ***end if***
12: $if\ index\ \leq\ sortedResourceList.Count\ or\ sortedResourceList[index] \geq jobResReq$;
13; ***return*** $index$;
14: ***end if***
15: ***end while***
16: ***end*** $Search$

4.6.4 Agent-Based Close to File Job Placement Mechanism

The idea behind the job placement algorithm known as the Close-to-File (CF) placement algorithm, was first discussed in (Mohamed and Epema, 2004). The function of the CF algorithm is to find a suitable set of execution sites for all submitted job components in a multi-cluster computing environment, and also find suitable file sites for the input file. The most important consideration here is of course, finding execution sites with enough processors and when there are options of choosing a site among set of suitable execution sites, selection is made such that the (estimated) delay of transferring the input file to the execution site is minimal. Shown in algorithm listing 10 is the pseudocode for the CF job placement algorithm.

Algorithm 10: Pseudocode for the Close-to-File job placement algorithm

1: $order\ job\ components\ according\ to\ decreasing\ size$
2: $\boldsymbol{for}\ each\ (job\ component\ j)\ \boldsymbol{do}$
3: $S_j = set\ of\ potential\ execution\ sites$
4: $\boldsymbol{if}\ (S_j \neq \emptyset)\boldsymbol{then}$
5: $select\ an\ E \in S_j$ // select the first
6: $\boldsymbol{else}$
7: $P_j = set\ of\ potential\ pair\ of\ execution\ site, file\ site$
8: $\boldsymbol{if}\ (P_j \neq \emptyset)\boldsymbol{then}$
9: $\boldsymbol{for}\ each\ \big((E,F) \in P_j\big)\ \boldsymbol{do}$
10: $estimate\ the\ file\ transfer\ time\ T_{(E,F)}$
11: $select\ the\ pair(E,F) \in P_j\ with\ minimal\ T_{(E,F)}$
12: $\boldsymbol{for}\ each\ (file\ site\ F'\ of\ the\ job)\ \boldsymbol{do}$
13: $insert(E,F)\ into\ the\ history\ table\ H$
14: $\boldsymbol{else}\ job\ submission\ fails$

Improving on this algorithm, some additional factors that might contribute to the choices of selecting the most qualified execution site or clusters can be considered, for any job components. Among the factors considered, are ranking the potential execution sites based on their processing capability such as speed, memory size, cache size,

storage capacity and network bandwidth. Shown in algorithm listing 11 is the Improved Close-to-File job placement algorithm (ICF).

Algorithm 11: Pseudocode for the Improved Close-to-File job placement algorithm (ICF)

1: $\boldsymbol{ICF}(CL_{ranked}, jobComponent_{new})$
2: $sort(jobComponent_{new})$*// order job components according to decreasing size*
3: $FS \leftarrow FileSize$ *{get the size of the input file}*
4: $BW \leftarrow Bandwidth$ *{get the file transfer rate between the execution site & the file site}*
5: $\boldsymbol{for}\ \forall\ jobComponent_j \in jobComponent_{new}\ \boldsymbol{do}$
6: $CL_c'' \leftarrow \{CL_1', CL_2', \dots, CL_c'\}\ set\ of\ potential\ execution\ sites\ (or\ ranked\ \ clusters)$
7: $\boldsymbol{if}\ CL_c'' \neq \emptyset\ \boldsymbol{then}$
8: $\boldsymbol{select}\ an\ CL_i' \in CL_c''$
9: $\boldsymbol{else}$
10: $P_j \leftarrow \{(CL_1', F_1), (CL_2', F_2), \dots, (CL_m', F_n)\}$ *// potential pairs of ranked execution sites, file sites*
11: $\boldsymbol{end\ if}$
12: $\boldsymbol{if}\ P_j \neq \emptyset\ \boldsymbol{then}$
13: $\boldsymbol{for}\ \forall\ (CL_m', F_n) \in P_j\ \boldsymbol{do}$
14: $FTT(CL_m', F_n) = \frac{FS}{BW}$ *{estimate the file transfer time between the execution site & file site}*
15: $select\ the\ pair\ (CL_m', F_n) \in P_j\ with\ minimal\ FTT(CL_m', F_n)$
16: $\boldsymbol{for}\ \forall\ F_n' \in job_{new}\ \boldsymbol{do}$ *// while F_n' denote a file site*
17: $insert\ (CL_m', F_n')\ into\ the\ history\ table\ H$
18: $\boldsymbol{else}\ job\ placement\ fails$
19: $\boldsymbol{end\ for}$
20: $\boldsymbol{return}\ (best\ selected\ pair\ (CL_m', F_n))$
20: $\boldsymbol{end\ for}$
21: $\boldsymbol{end\ if}$
22: $\boldsymbol{end\ for}$

The Improved-Close-to-File algorithm uses the following parameters:

a. The numbers of qualified ranked nodes in the sites of a grid: A job component can only be placed on an execution site in which the site node resource availability is either greater or equal to the job component resource requirement.

b. The file size: The size of the input file, which enters in the estimates of the file transfer times.

c. The network bandwidths: The bandwidth between an input file site and an execution site, gives the opportunity to estimate the transfer time of a file given its size. Therefore,

there is need to measure and forecast the network bandwidth when selecting the execution sites, in order to minimize the file transfer times.

ICF algorithm would usually maintain a history table H (Fig. 4.29) with a subset of pairs of potential execution site and input job file sites to consider. ICF selects from H all potential pairs (CL'_m, F_n) of execution site, file site, with CL'_m having a sufficient number of qualified ranked execution nodes for the job component and F_n being a file site of the Job (line 10). However, if no such pair exist in H, then the job component, and hence the whole job, currently cannot be placed (line 18). Otherwise, ICF estimates for each selected pair the File Transfer Time (FTT) from the file site to the execution site (line 14), and picks the pair with the lowest estimate (line 15). If (CL'_m, F) is the pair selected, ICF insert into H all pairs (CL'_m, F') with F' a file site of the job (lines 16, 17). Note that if the history table is initially empty, it will remain empty. Therefore, it has to be initialized with some set of suitable pairs of execution and file sites.

Several of the important components of network performance can roughly be estimated using the file transfer rate formula. For instance, if the file size of an application is known, dividing the file size by the network bandwidth yields an estimate of the fastest time that the file can be transferred. Therefore, the general formula for computing an estimate of the FTT as used in this dissertation is given as follow:

$$FTT(s) = \frac{FS(mb)}{BW(mbps)} \tag{4.34}$$

where FTT is the estimated file transfer time

FS is the input file size

BW is the network bandwidth between the execution site and the input file

Two important points should be considered when doing this calculation.

a. The result is an estimate only, because the file size does not include any overhead added by encapsulation.

b. The result is likely to be a best-case transfer time, because available bandwidth is almost never at the theoretical maximum for the network type. A more accurate estimate can be attained if throughput is substituted for bandwidth in the equation.

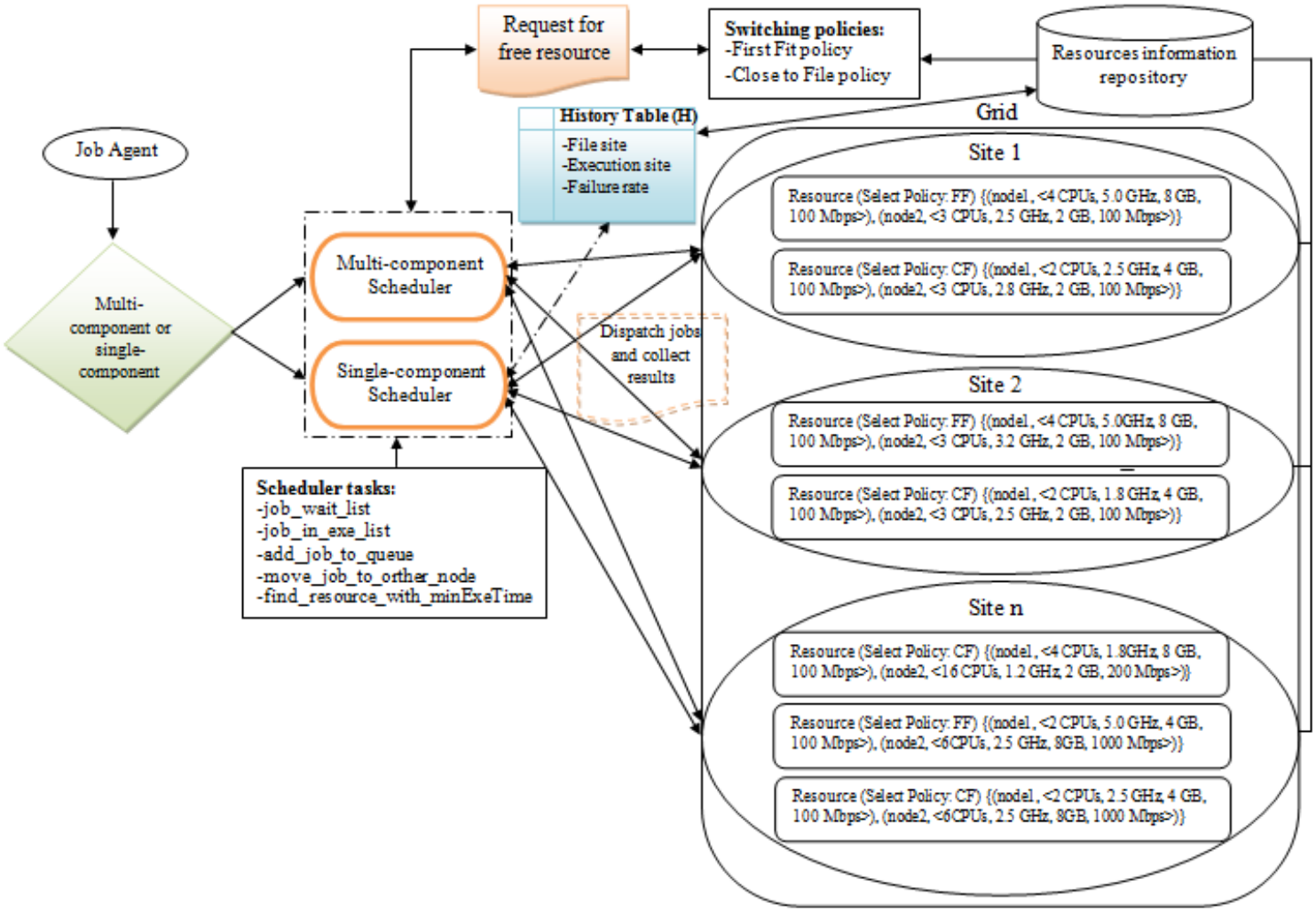


Fig. 4.29: A use case model for agent-based scheduling design pattern, starting from job submission stage to resource allocation stage.

4.6.5 Agent-Based Job-Resource Assignment Model

In the proposed scheduling system, submitted jobs, job_i are queued in a prioritized global queue, where $i \in \{1, 2, \dots, l\}$ and l is the total number of available job in the global queue, and cluster resources are denoted by $r_{j,k}$, where $j \in \{1, 2, \dots, m\}$ and m is the number of resource per node. $k \in \{1, 2, \dots, n\}$ and n is the number of cluster. The agent based job to resource match making is started when job is passed on to the scheduler by the job agent who picks up job_i from the global queue based on the FCFS mechanism. The match making agent which is part of the scheduling agent extracts job resource components information, from the submitted job and forwards same to suitable resource selector agent. The suitable resource selector agent needs to calculate the rank of cluster resource(s) based on available processing elements resource characteristics, for each of the suitable and available clusters.

The Matchmaking Agent uses the following selection model in assigning jobs to resource(s):

a. Get and evaluate the resource requirement of the job.
b. Identify suitable cluster(s) based on information from (i) above.
c. Compute the rank and node reliability of each machine per cluster using equation shown below.
d. Select clusters based on nodes with the highest ranked resource quality and reliability level.

Algorithm listing 12 presents the selection criterion for the ranked clusters:

Algorithm 12: Pseudo-code for resource selection model

```
SModel (CL_i,CL_j ){
if ((Rank(CL_i) > Rank(CL_j)) && (NR(CL_i) > NR(CL_j) ))
        {
                selectedClusters ← select (cluster_i)
        }
              else if (NR(CL_i) < NR(CL_j))
                {
                        selectedClusters ← select (cluster_j)
                }
                else
                        {
                                selectedClusters ← select (cluster_j)
                        }
           return selectedClusters
        }
```

Harmonising the resource scheduling framework, algorithm listing 13, shows the proposed agent-based matchmaking resource allocation algorithm. It can be recalled, that the initial issue in the efficient selection of resources is matching the right resource(s) with the right job. The main focus of this dissertation has always been to model an architectural design pattern that broadly deals with the issues of efficient resource selection and matching of users' jobs to most suitable and efficient selected resources. Shown in algorithm listing 13 is the pseudocode for the proposed Agent-based Matchmaking Scheduling Algorithm.

Algorithm 13: Pseudo-code for Agent-based Matchmaking Algorithm

1: ***Data***: $jobQueue \leftarrow \{job_1, job_2, \dots, job_i\}$ //jobQueue consists of jobs, job_i, where $i = 1, \dots, l$;
$CL_c \leftarrow \{machine_{1,c}, machine_{2,c}, \dots, machine_{j,c}\}$
$machine_{j,c}$ are $available$ $nodes$ $resource(s)per$ $cluster$ c where $j \in$
$1, \dots, k$ $denote$ $resource$ $index$; and c $1, \dots, C$ $denotes$ $cluster$ $index$;
$SModel$ is the selection model for ranked cluster resource
$C = 1$; initialises the number of cluster to 1.
2: ***Result***: job_i and $machine_{j,c}$ $assignments$
3: ***Begin***
4: ***while*** $jobQueue \neq \emptyset$ ***do***
5: ***for*** $\forall\, job_i \in jobQueue$ ***do***
6: $job_{new}[i] \leftarrow indexJob(job_{initial}[i])as\ the\ first\ job\ in\ the\ queue$; // done by job Agent
7: $jobResReq \leftarrow get$JobResourceInfo in $job_i \in jobQueue$;
8: $CL'_c \leftarrow search(CL_c, jobResReq)$; {get list of suitable resource from resource provider}
9: ***for*** $\forall\, machine_{i,p} \in CL'_p,\ machine_{j,q} \in CL'_q$ ***do***
10: $CL''_{selected} \leftarrow SModel(CL'_p, CL'_q)$; // PF is the cluster performance function
11: ***if*** $C > 1$ ***then***
12: $Assignment_i \leftarrow ABFF(job_{new}, CL''_{selected})$;
13: ***else***
14: $Assignment_i \leftarrow CFJP(job_{new}, CL''_{selected})$; // Close to file job placement algorithm
15: ***end if***
16: ***if*** $Assignment_i$ is better than $Assignment_j$ ***then***
17: $Assignment_j \leftarrow Assignment_i$;
18: ***end if***
19: ***end for***
20: ***end for***
21: ***execute*** $Assignment_j$;
22: $removeFromQueue(Assignment_j)$;
23: ***end while***
24: ***end***

The scheduling algorithm described above is used in the implementation of the general-purpose scheduling framework. The complexity of the algorithm is determined by the number of possible selections of highly ranked quality clusters:

$$C_n^1 + C_n^2 + \cdots + C_n^n = 2^n - 1 \qquad (4.35)$$

It is therefore essential that the three evaluation processes (goodness function, performance function and resource ranking) are efficient. During each scheduling process, the evaluation functions can be called $2^n - 1$ times. Even in the situation where all the processors of a distributed resource are homogeneous, the evaluation

functions still need to be called n time. The evaluation can be performed very quickly to produce prediction results on the fly; this is a key feature of the Agent-Based Matchmaker which enables the integration of the different evaluation models, presented in this dissertation to provide service discovery support for the general-purpose scheduling framework.

4.7 Main Contributions of Chapter

In this chapter, the need for exploiting learning using neural networks is expressed for the general-purpose of efficient scheduling of user's application, and selection of best candidate resources, as against the conventional methods of scheduling. Then the resource selection problem is formulated, and explained the limitations of conventional scheduling algorithms. We pointed out that learning should be applied to achieve transparency as well as adaptability in dynamic and uncertain distributed resource environment. ANN was employed as the learning algorithm due to its fast training speed and satisfactory generalization performance. After introducing the training process of ANN, the Agent-based intelligent resource selection algorithm is proposed. Simulation experiments showed that Agent had the capability of predicting relative node performance with small errors. Scheduling simulations were conducted to compare the performance of conventional and proposed algorithm, and the results indicated that Agent-based resource selection outperformed the conventional algorithm, achieving higher computing power utilization.

The training of Agent is currently based on supervised learning, and thus training data needs to be labelled. However, for real-world Grid applications, labelled data may not be available all the time. Incorporating reinforcement learning with agent networks will allow the learning to be performed through continued interaction with the Grid

environment and therefore, this method is recommended. Therefore, providing proposed model with reinforcement learning will be studied in the future.

CHAPTER FIVE

FRAMEWORK EVALUATION

5.1 Synopsis

In this chapter, the suitability and the performance of the proposed approach is evaluated through several experiments. The goal is to demonstrate how the applied techniques are able to deal with various application scenarios. To be more precise, the evaluation demonstrates how the proposed scheduling techniques perform when different problems are considered. For this purpose, different data sets were used during the experiments. All such experiments with exception of the first experiment were performed using the Alea Grid simulator (Klusáček, 2010) with an extension of Agent Repast (Isaac, 2014) and GridSim simulator (Sulistio *et al.*, 2008).

This chapter is organized as follows: first, a description of the data set that was used to evaluate the proposed techniques is presented. This is followed by a detailed description of the experimental setup, the experimental test bed, applied optimization criteria and scheduling algorithms. Four major experiments follow, demonstrating the performance of the envisioned solution.

In the first experiment, critical study and analyses of the impact of various resource configuration parameters on the performance of a computational cluster was conducted. The intent is to determine certain resource configuration specifications parameters that are responsible for selecting cluster resources, appropriate for a particular problem run based on that run's characteristics. In test 1, both single and combined effects of four resource configuration parameters were considered, number of processor per node, CPU speed, memory size, and cache size. In test 2, similar experiment was repeated, but cache size is replaced with associated network bandwidth. In the experiment, a

measurement-based evaluation technique was used to characterize the specific performance contribution, of the individual Grid resource configuration parameters. In the process, the key primary parameters (or factors) that should be considered when selecting and allocating a computational node for user application execution were identified.

The second experiment, demonstrates the application of the proposed scheduling solution in a multi-agent environment. In this test case, MAS is used to perform scheduling functions for several classes of multi-component applications (applications consisting of multiple schedulable components). The scheduling of multiple interacting components across multiple sites is a complex problem, especially when the available resources and specifically network capacity are assumed to vary between sites. In a second test case, a scheduling mechanism that uses routing indices and intelligent agents in each node, to perform global scheduling in a collaborative and coordinated manner is presented. The behaviour of the proposed scheduling model was analysed and compared with three other scheduling algorithms, in terms of selecting the best execution site that is most appropriate for specific application characteristics.

The last experiment demonstrates the analysis and performance evaluation of the two proposed scheduling policies, that pertains to job scheduling and resource allocation. The two scheduling policies include; the Agent-based First-Fit Resource Selection Policy and the Agent-Based Close to File Job Placement Policy, discussed in section 4.3.3 and 4.6.4. These scheduling policies treat different multi-component applications requiring diverse heterogeneous resources fairly. The policies are used by MAS to schedule user applications and to also find available and qualified distributed resource that are capable of executing user application at a very minimal time.

5.2 Data Set

In this section, the simulation configuration parameters used to evaluate the proposed scheduling policies and matchmaking algorithm discussed in this dissertation, is presented. The data set was downloaded from the workload logs of real systems as explained in the next section. See appendix for the sample data sets.

5.3 Data Sets from MetaCentrum

MetaCentrum is the Czech National Grid Infrastructure (metaCentrum, 2014). It coordinates and operates the computational and storage resources in the Czech Republic. With permission, the existing workload logs were used to prepare two real life-based data sets. The data set MetaCentrum'09 is based on the data collected during the first five months of 2009. MetaCentrum'09 contains a trace of 103, 620 jobs. The specific job requirements (of available CPU being greater than or equal to user CPU requirements), are included in the data sets. Besides that, the detailed clusters' descriptions with the information about machine architecture, CPU speed, memory size and the supported properties are provided. MetaCentrum'09 represents 14 clusters (806 CPUs). Also, the list of available queues including their priorities and associated time limits is provided. Moreover, the trace of machine failures and the description of machines temporarily dedicated for special purposes are provided as well in the appendix section of the Dissertation.

5.4. Cluster Performance Analysis (Experiment 1)

The optimization of distributed resources, through designing and modelling of computer experiments is becoming very popular in scientific applications (Chen *et al.*, 2003). Depending on the objective, different design and modelling tools may be needed to efficiently sample and accurately approximate the outputs of the system. In general,

there are no universally optimal tools for either task. Nevertheless, under a particular application objectives and specific structures of the input-output relationships, some tools can be shown to be better than the others. It will be of great interest to develop and identify appropriate classes of design and modelling tools for various types of applications (Douglas, 2008).

Following the combinatorial derivation of the characterization process derived earlier on, which was obtained as 2^k factorial, the cluster analysis experiment is based on the 2-k factorial design model to help us identify some of the desired factors required to rate the performance of a specific cluster. A full factorial design experiment is made up of a number of factors or parameters, each with a number of levels as in our case, low and high, with all combinations of the levels being performed. The main effects of each of the parameters on the response can be determined, as well as the two-way interaction effects of each of the pairs of factors, and higher-order interaction terms.

One instance of the application of the experimental design technique is the analysis of cluster performance, using different resources configuration characteristics or factors, where factors are binary (e.g. a lower value and a higher value). Let's consider the following factors say *A, B, C,…,* and the output of interest y, then the output *y* may be represented as:

$$\begin{aligned} y = q_0 + \{q_A x_A + q_B x_B + q_C x_C + \ldots\} + \\ \{q_{AB} x_A x_B + q_{AC} x_A x_C + q_{BC} x_B x_C + \ldots\} + \\ \left\{q_{ABC} x_A x_B x_C + \ldots\right\} + \ldots, \end{aligned} \tag{5.1}$$

Here X_A is *-1* when factor *A* takes low value and *+1* when it takes high value, same for X_B, X_C, . . . are *-1* or *+1* corresponding to whether *B,C,…* are low or high. q_0 is the mean effect, the q_i are the main effects, $q_{i,,j}$ are the two-way interaction effects, and $q_{i,j,k}$ and so

on, are the higher-order interaction effects. Where $i = A, B, \ldots,$ and $i,j = AB, AC, \ldots, i,j,k = ABC, ABD, \ldots$

In next section, some experimental scenarios (two examples) were presented to determine the effects of the different combinations of grid cluster resource configurations. In section six, a discussion based on simulation result, the performance of a cluster in respect to the percentage contributions and interactions of four configuration parameters, earlier on mentioned, are presented. In this experiment, only instantaneous resource availability was considered for the resource characterisation process.

5.4.1 Applications

Coarse grids (2^k factorials), are most efficient if assumed that the simulation response, is well-fit by a model with only linear main effects and interactions, while fine grids (more than two levels for factors), provide greater detail about the response and greater flexibility for constructing meta-models of the responses (Mee, 2009). Next, an example is given to illustrate how the factorial design can be applied to analyse the impact of some selected cluster resource specification as shown in Table 2 and 3.

Example: The 2^K Factorial Design (k = 4)

In this section, the impact of memory size and number of processor on the performance of a computational cluster is studied. The intent is to determine certain resource configuration specifications, responsible for selecting cluster resources, appropriate for a particular problem run based on that run's characteristics. In this example, both single and combined effects of four resource configuration specifications are considered; number of processor per node, CPU processing speed, the memory attached to the processor on resource prioritization, and cache size. Each resource characteristic is

referred to as a factor. A resource can have several other factors such as storage capacity, network bandwidth, and so on. A factor is usually classified into two levels of priorities *H (+)* and *L (-)* as mentioned earlier in section IV.

In this example, a study of the impact of some resource specifications on the performance of a computational grid cluster and its prioritization (Table 5.1 and 5.2) is presented.

Table 5.1: Factors and Levels Configuration

	Factor	**Level 1**	**Level 2**
A	Memory size	8MB	64MB
B	Cache Size	2KB	4KB
C	Processor Speed	5GHz	10 GHz
D	Number of processor	6	12

Table 5.1 consist of sample factors to be analysed such as memory size, processor speed, cache size, and number of processors labeled *A, B, C, and D.* Level 1 and level 2 in the Table represent the resource configuration quality, whether low (Level 1) or high (Level 2) in terms of specifications.

Table 5.2: A 2^{k-1} Design

Cache size	**8MB**		**64MB**	
	6	12	6	12
2KB	16	48	24	60
4KB	12	52	36	88

From equation (5.1), the performance of this cluster as a regression model is expressed as follows:

$$y = q_0 + q_A x_A + q_B x_B + q_C x_C + q_D x_D + q_{AB} x_A x_B + q_{AC} x_A x_C + q_{AD} x_A x_D + q_{BC} x_B x_C + q_{BD} x_B x_D + q_{CD} x_C x_D + q_{ABC} x_A x_B x_C + q_{ABD} x_A x_B x_D + q_{ACD} x_A x_C x_D + q_{BCD} x_B x_C x_D + q_{ABCD} x_A x_B x_C x_D \quad (5.2)$$

$$x_A = \begin{cases} -1 \ \text{if } 8MB \text{ memory} \\ 1 \ \text{if } 64MB \text{ memeory} \end{cases}, x_B = \begin{cases} -1 \ \text{if } 2KB \text{ cache} \\ 1 \ \text{if } 4KB \text{ cache} \end{cases}$$

$$x_C = \begin{cases} -1 \ \text{if processor speed is } 5\text{GHz} \\ 1 \ \text{if processor speed is } 10\text{GHz} \end{cases}, x_D = \begin{cases} -1 \ \text{if } 6 \text{ processor} \\ 1 \ \text{if } 12 \text{ processor} \end{cases}$$

Determining the importance of a factor:

a. Importance of a performance factor: Variation of Factor/ Total variation
b. calculating sample variance of $y =$

$$s_y^2 = \frac{\sum_{i=1}^{2^4} \left(y_i - \bar{y}\right)^2}{2^4 - 1} \quad (5.3)$$

c. calculating total variation or sum of squares total (SST) of y:

$$y = SST = \sum_{i=1}^{2^4} (y_i - \bar{y})^2 \quad (5.4)$$

SST can be transformed to

$$SST = 2^4 q_A + 2^4 q_B + 2^4 q_C + 2^4 q_D + 2^4 q_{AB} + 2^4 q_{AC} + 2^4 q_{AD} + 2^4 q_{BC} + 2^4 q_{BD} + 2^4 q_{CD} + 2^4 q_{ABC} + 2^4 q_{ABD} + 2^4 q_{ACD} + 2^4 q_{BCD} + 2^4 q_{ABCD} \quad (5.5)$$

These three parts represent the portion of the total variation explained by the effects of A, B,C, D and the interactions *AB, AC, AD, BC, BD, CD, ABC, ABD, ACD, BCD,* and *ABCD*. See Appendix D for a simplified transformation example of this model.

The sign Table method (in Table 5.3) is introduced here to compute the effects of the different factors; sign Table simply contains the effects of all factors in question. The generator uses the fact that squaring the entries in any given column gives a column of ones, which can be thought of as an identity element I. If the *A* effect with the *BC* interaction is confounded, then that is equivalent to declaring $A * BC = ABC = I$. It follows that $B = BI = B * ABC = AC$, so *B* is confounded with the *AC* interaction. Similarly, *C* is confounded or aliased with *AB*, and the overall mean (I) is confounded or aliased with *ABC*.

However, $y\ or\ SST$ is computed as follows:

$$SST = 2^2 q_A^2 + 2^2 q_B^2 + 2^2 q_{AB}^2 + \cdots + 2^2 q_{ABCD}^2$$

$$SST = 2^4(7.88^2 + 2.88^2 + 4.13^2 + 24.88^2 + 6.63^2 + (-1.88)^2 + 3.88^2 + 3.63^2 + (-2.13)^2 + (-2.13)^2 + (-0.88)^2 + 4.88^2 + 4.63^2 + (-4.88)^2 + 2.88^2) = 13910.26$$

SSA/SST = 16(62.09)/13910.26 = 7.14%, SSB/SST = 16(8.29)/13910.26 = 0.95%

SSAB/SST = 16(17.06)/13910.26 = 1.96%, SSC/SST = 16(619.01)/13910.26 = 71.20%

SSAC/SST = 16(43.96)/13910.26 = 5.06%, SSBC/SST = 16(3.53)/13910.26 = 0.41%

SSABC/SST = 16(15.05)/13910.26 =1.73 %, SSD/SST = 16(13.18)/13910.26 =1.52 %

SSAD/SST = 16(4.54)/13910.26 = 0.52%, SSBD/SST = 16(4.54)/13910.26 =0.52 %

SSABD/SST = 16(0.77)/13910.26 = 0.09%, SSCD/SST = 16(23.81)/13910.26 = 2.74%

SSACD/SST = 16(21.44)/13910.26 = 2.47%, SSBCD/SST = 16(23.81)/13910.26 = 2.74%

SSABCD/SST = 16(8.29)/13910.26 = 0.95%.

The sign Table method (in Table 5.3) is introduced here to compute the effects of the different factors; sign Table simply contains the effects of all factors in question. The generator uses the fact that squaring the entries in any given column gives a column of ones, which can be thought of as an identity element I. If the *A* effect with the *BC*

interaction is confounded, then that is equivalent to declaring $A * BC = ABC = I$. It follows that $B = BI = B * ABC = AC$, so B is confounded with the AC interaction. Similarly, C is confounded or aliased with AB, and the overall mean (I) is confounded or aliased with ABC.

Table 5.3: Sign Table Method to Determine the Effects of Factors for *K=4*

FACTORS	I	A	B	AB	C	AC	BC	ABC	D	AD	BD	ABD	CD	ACD	BCD	ABCD	Y
	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	16
	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	24
	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	12
	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	36
	1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	48
	1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	60
	1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	52
	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	88
	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	14
Effect of factors	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	10
	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	36
	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	18
	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	78
	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	92
	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	46
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	100
Total Variation	730	126	46	66	398	106	-30	62	58	-34	-34	-14	78	74	-78	46	730
Estimate of effects of factors	45.62	7.88	2.88	4.13	24.88	6.63	-1.88	3.88	3.63	-2.13	-2.13	-0.88	4.88	4.63	-4.88	2.88	Total/2^4
SST(%)		7.14	0.95	1.96	71.2	5.06	0.41	1.73	1.52	0.52	0.52	0.09	2.74	2.47	2.74	0.95	100%

Table 5.3 suggests that the effects of resource component *A* (memory) and resource component *C* (processor speed) are the largest. The linear effect of resource factor *C* is the most significant, with an estimate of nine times the estimate of the next most significant resource factor, *A*; showing that resource component *C* is very significant and important relative to the other resource components. Both factors account for 78.34 % of the total sum of squares in the data. Aggregate resource factor composition, the four main effects contribute 80.81% of the sum of squares, six two-factor interactions contribute 11.21%, the four three-factor interactions contribute 7.03%, and the one four-factor interactions contribute 0.95%.

5.4.2 Simulation Results and Discussion

The experiment design selected in this case was a four factor, two interaction design. A standard Yates Algorithm Design (Yates and Cochran, 1938) with 8 experiments runs (Table 5) was used. In this simulation, only two-factor interactions A × B and A × C was studied. The interactions have a common factor between them. That is the limitation of this design because of the condensed sample space. This is the only way we can study two interactions with this design, without mistaking main effects with interaction effects. Table 5.4 contains the factors name and level settings, while Table 5.5 contains the composition of the factors common interactions. And finally, a complete analysis of variance (ANOVA) was carried out, to obtain the summary of result shown in Tables 5.9 and 5.10.

Experiment parameters such as varying cluster node specifications, were accessed from selected cluster resource information, before experiment samples were taken. The resource configuration information selected included varying node ram size, cache size, processor speed and the associated bandwidths. In this design with 8 runs, each factor

has 4 runs dedicated to each level setting for a factor. 5 samples x 4 runs = 20 data points per level of a factor which is a good minimum sample size. Generally, 20 to 25 data points per level setting of a factor were considered appropriate for the experiment. Discovery and selection of best candidate resources were tested under the experiment design conditions based upon the experiment run.

The data is entered into the columns corresponding to the experiment run as shown in Table 5.6. The mean and the Log(s) are calculated above the data entry fields (Table 5.4, 5.5). An analysis of variance, ANOVA, is conducted for assessing which factors effect centering (means analysis) and which factors effect the variation (Log(s) Analysis).

Table 5.4: Factors and Level Settings

FACTORS	NAME	LEVEL 1	LEVEL 2
A	Processor Speed	3 GHz	5 GHz
B	Network Bandwidth	100Mbps	150Mbps
C	Memory Size	4 GB	5 GB
D	Number of Processor	15	25

Table 5.5: Factors Common Interaction

Interactions	Factor 1	Factor 2
A×B	Processor Speed	Network Bandwidth
A×C	Processor Speed	Memory Size

Table 5.6 illustrates the data entry from our experiment, which is presented in the column that represents the experiment run. The performance statistics are calculated from the given data. Mean for centering and Log(s) for variation.

Table 5.6: Experiment Data Entry Sheet

Perf Stat	Run #1	Run #2	Run #3	Run #4	Run #5	Run #6	Run #7	Run #8
Mean	20.317	17.833	12.533	19.017	18.633	18.715	15.483	22.417
Log(s)	7.639	-3.105	5.213	1.032	3.509	1.586	-5.195	5.336
Sample Number	Run #1	Run #2	Run #3	Run #4	Run #5	Run #6	Run #7	Run #8
1	20.1	15.6	12.3	18.7	18.9	17.8	18.7	22.2
2	20.3	19.2	12.6	18.8	18.1	19.8	10.2	22.1
3	20.6	19.1	12.8	18.9	19.3	18.4	15.7	22.4
4	20.2	15.5	12.5	18.6	18.7	19.2	13.9	22.6
5	20.4	17.2	12.1	20.6	18.6	18.51	19.2	22.3
6	20.3	20.4	12.9	18.5	18.2	18.58	15.2	22.9

Tables 5.7 and 5.8 present the means analysis for assessment of centering of the response distribution. The data is organized by factor and level setting. The setting averages are calculated now for use after the ANOVA determines which factors are significant. The graphs (Figs.5.1 - Fig.5.3) help decide the best level setting, which is to either minimize or maximize the resource configuration characteristic. The Sum of Squares values are used in the ANOVA.

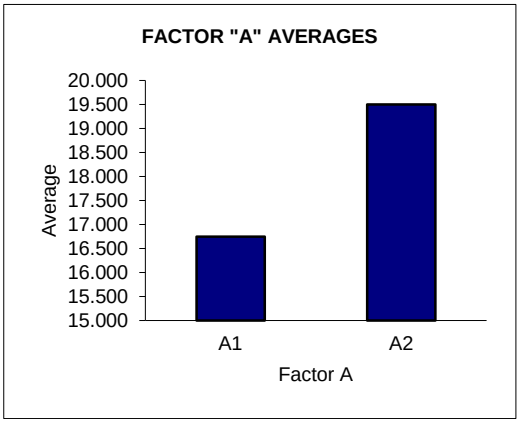
Table 5.7: Analysis of Averages

Column	Level	Factor	Setting	Setting Sum	Setting Average	Sum of Squares
A	A1	Processor Speed	3 GHz	66.97	16.742	15.166
	A2		5 GHz	77.98	19.495	
B	B1	Network Bandwidth	100Mbps	75.50	18.875	4.573
	B2		150Mbps	69.45	17.363	
C	C1	Memory Size	4 GB	69.70	17.425	3.848
	C2		5 GB	75.25	18.812	
D	D1	Number of Processor	15	73.53	18.383	0.559
	D2		25	71.42	17.854	
E	AxB-1	PS×NB-1	1	64.57	16.141	31.277
	AxB-2	PS×NB-2	2	80.38	20.096	
F	AxC-1	PS×MS-1	1	70.97	17.742	0.000
	AxC-2	PS×MS-2	2	70.97	17.742	
					Total Sum of Squares =	62.953
					Overall Average of the Means =	18.119

PS-Process speed, NB-Network bandwidth, MS- Memory size

Table 5.8: Experiment Data and Design Layout

									Setting	Setting
Perf Stat	**Mean**	A	B	C	D	AxB	AxC	Interactions	Sum	Average
Run #1	**20.317**	1	1	1	1	2	2	A1-B1	38.950	19.475
Run #2	**17.833**	2	1	1	2	1	1	A1-B2	28.017	14.00833
Run #3	**12.533**	1	2	1	2	1	2	A2-B1	36.548	18.274
Run #4	**19.017**	2	2	1	1	2	1	A2-B2	41.433	20.717
Run #5	**18.633**	1	1	2	2	2	1			
Run #6	**18.715**	2	1	2	1	1	2	A1-C1	32.850	16.425
Run #7	**15.483**	1	2	2	1	1	1	A1-C2	34.117	17.058
Run #8	**22.417**	2	2	2	2	2	2	A2-C1	36.850	18.425
								A2-C2	41.132	20.566



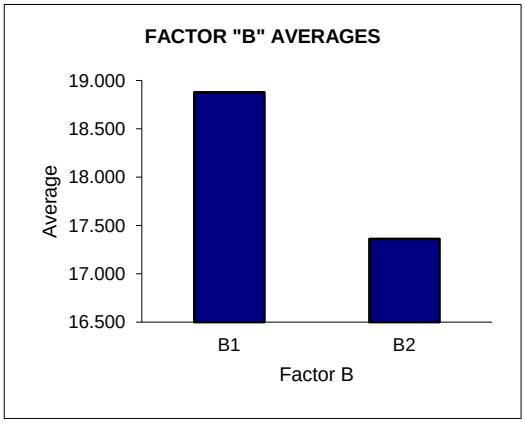
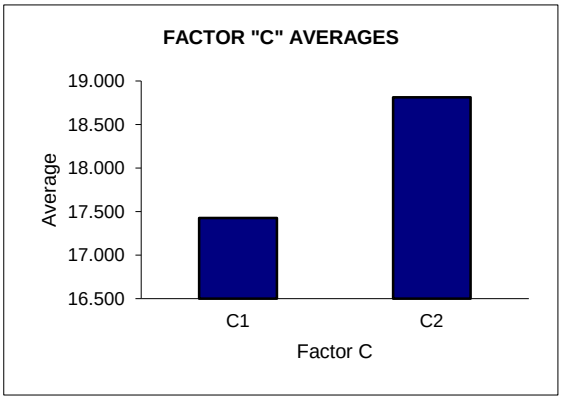


Fig.5.1: Analysis of Effects of Factors A (left) and B (right)



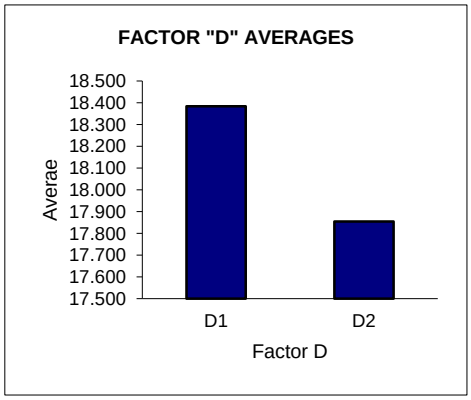
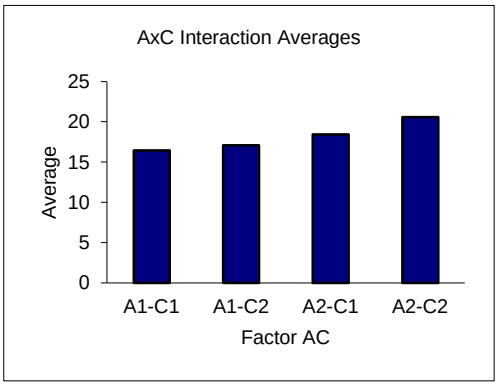


Fig.5.2: Analysis of Effects of Factors C (left) and D (right)



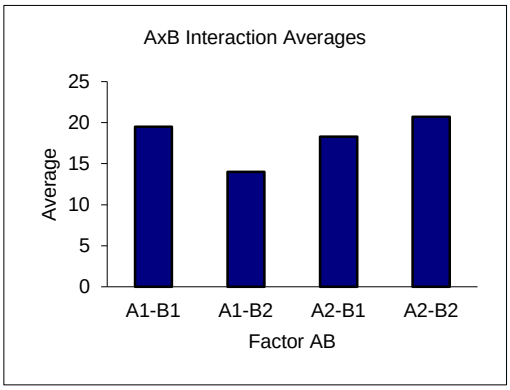


Fig.5.3: Interaction Between Factors AB (left) and AC (right)

In Table 5.9, if the value of the F-Ratio statistic test is less than or equal to one, then it must be pooled into the unexplained, or error. The "yes" and "no" under the column headed "POOL??" in Table 5.9, indicates whether a factor should be pooled or should

not be pooled. The results are tabulated in the final ANOVA Table 5.10. If after pooling the F-Ratios less than or equal to one, there should be about half the columns left, six was started with, so there should be three to four columns left after pooling. However, in the case of inadequate pooling the next smallest F-Ratio can be used until there is at least three or four columns left.

Table 5.9: Initial Anova Table - Prior to Pooling

COLUMN	SOURCE	df	SS	MS	F-Ratio	Prob >F	POOL??
A	Processor Speed	1	15.166278	15.1663	**2.01**	0.3908	**No**
B	Network Bandwidth	1	4.572792	4.5728	**0.61**	0.5786	**No**
C	Memory Size	1	3.848000	3.8480	**0.51**	0.6049	**No**
D	Number of Processor	1	0.559153	0.5592	**0.07**	0.8306	**Yes**
E	AxB Interaction	1	31.277459	31.2775	**4.15**	0.2904	**No**
F	AxC Interaction	1	0.000000	0.0000	**0.00**	1.0000	**Yes**
	Error	**1**	**7.529570**	**7.529570**			
	Totals	7	62.953252				

Significant factors have Prob>F values of 0.05 or less (equates to confidence of 95% or greater) and percent contributions greater than 5%. The unexplained, or error, Contribution % was considered to be 50% or less for this type of design of experiment to be valid. Between 50% and 65% caution is required with the results, greater than 65% is a poor experiment and the results are highly questionable. In this situation the experiment needs to be repeated all over again. If the means shows little and Log(s) shows a lot then the factors affected the spread more than centering - or vice versa (Table 5.10).

Table 5.10: Final Anova Table - Prior to Pooling

COLUMN	SOURCE	df	SS	MS	F-Ratio	Prob >F	PURE SS	CONTRIB. %
A	Processor Speed	1	15.166	15.17	5.62	**0.098**	12.47	**19.81%**
B	Network Bandwidth	1	4.573	4.57	1.70	**0.284**	1.88	**2.98%**
C	Memory Size	1	3.848	3.85	1.43	**0.318**	1.15	**1.83%**
D	Number of Processor							
E	AxB Interaction	1	31.277	31.277	11.60	**0.042**	28.58122	**45.40%**
F	AxC Interaction							
	Pooled Error	**3**	**8.089**	**2.696**			**18.87369**	**29.98%**
	Totals	7	62.953				62.95325	100%

F→Test Statistical Value: Prob→Probability Value

Table 5.10 suggests the effects of resource factor A (processors speed), B (network bandwidth), and C (memory attached to a processor) significantly. However, the linear effect of resource factor A outweighs the remaining two factors B and C by a ratio of 1:6:10. The linear effect of resource factor A is most significant with an estimate of six times the estimate of the next most significant factor, B and 10 times the estimate of factor C; showing that resource factor A contribute very significantly to the performance of a particular computing cluster resource. The three factors account for 24.62 % of the aggregate resource factor contribution. The one main two-factor (AXB) interactions contribute 45.40%, and the residual effect is 29.98%. The high percentage contribution of the two factor interaction effect reveals the relative importance of both processing speed and associated bandwidth to the performance of a cluster resource.

The experiment was simulated iteratively severally, with different input data set to identify those important resource characteristics or factors that must be considered, in order to decide the quality of an administrative domain or execution site to run a particular application. The experimental findings reveal that the most significant resource parameter with high impact factor after several run is the processing speed of a

resource, next to that in ranking is the processor associated bandwidth, and finally the memory capacity attached to processors. The graphs in Fig.5.4 (a) to (f) interpret the log(s) analysis for assessment of response distribution variation.

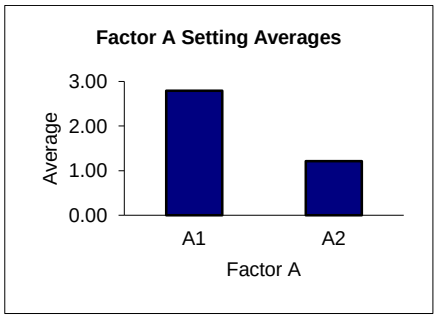


Fig.5.4a

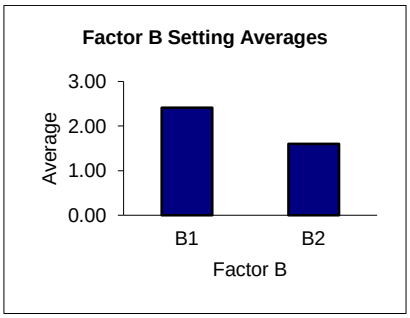


Fig.5.4b

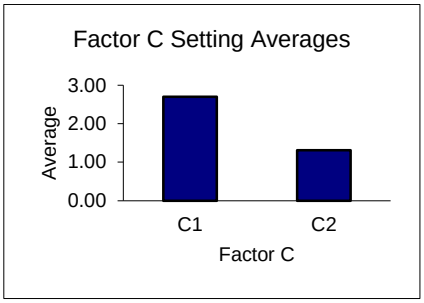


Fig.5.4c

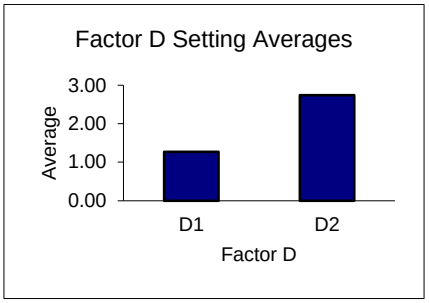


Fig.5.4d

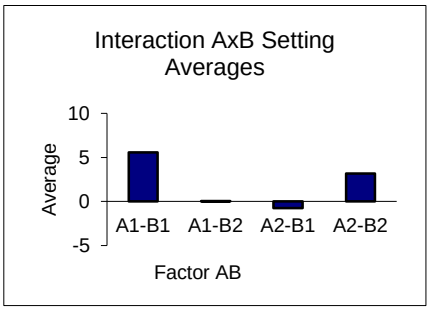


Fig.5.4e

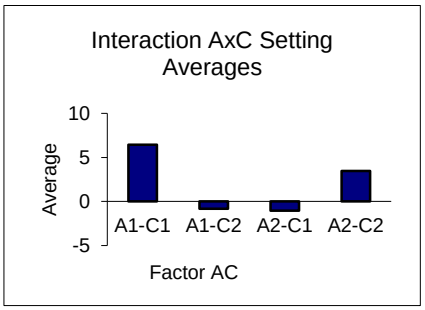


Fig.5.4f

Fig. 5.4: Assessment of Response Distribution Variation.

The conducted experiment has shown how design of experiments (DOEs) methodologies, such as factorial design can be extended to the domain of distributed

systems environment. DOEs in this case, is used as a systematic method to determine the relationship between factors affecting resource performance and their selections. In other words, it is used to find cause-and-effect relationships. This information is required to guide the scheduler in managing application resource requirements (inputs), in order to optimize the scheduling process (output). Using the factorial design method of measurement and evaluation technique, a measurement-based evaluation model for the characterization of computational Grid resources was presented successfully, in terms of the performance of CPU, main memory, interconnect and clusters.

By understanding the complexity of resource configuration characteristics and the interactions between the specific resource parameters, the experiment has shown how to identify optimal resource combination parameters, that offers minimal performance cost (execution time and utilization) with regards to specific application requirement using DOE. The experiment results have also shown that for every resource selection, three resource parameters (processor speed, associated bandwidth and memory size attached to each processor) and their interactions are the key primary factors that should be considered when allocating computational nodes to user application for optimal execution.

5.5 Simulation Using Agent Grid Repast Simulator (Experiment 2)

The experiments were conducted using an open source generic agent-based Grid simulator (Agent Grid Repast Simulator (Isaac *et al.*, 2007)), specifically built for developing agent coordination mechanism on top of Grid systems. The simulator focuses on the development of agents for coordination of Grid activities such as scheduling, load balancing virtual organisation management and others. It is built on top

of the repast agent simulation engine. Agent Grid Repast Simulator provides two types of models; Grid Model and Agent Framework Model.

The Grid Model simulates a Grid with sites hosting resources and a physical network of links between the sites. Grid activities such as job scheduling to the machines, job processing in the resources and data transfers over the links are simulated.

The Agent Framework Model simulates an agent layer on top of the Grid Model. It completely abstracts the Grid Model, shifting the focus to agent development. The framework facilitates the development of autonomous agents for the coordination of the Grid activities, as well as the learning/communication processes between the agents. Realistic values are chosen for the parameter ranges (Table 5.11).

Table 5.11: Simulation Parameters

Parameter name	Value range
Number of components	2, 4, ... 12
Number of sites	1, 2, ..., 15: [S1, S2 ... S15]
Inter link bandwidth rate	[0.5, 2] GBps
Inter link bandwidth rates between sites	S1-S2:(1), S1-S3: (1.5), S2-S4:(1.2), S2-S5:(1.5), S3-S6:(1.5), S3-S7:(1.5), S4-S8:(0.9), S4-S9:(1.5), S5-S10:(0.9), S5-S11:(1.5), S6-S12:(0.9), S6-S13:(1.5), S7-S14:(0.9), S7-S15:(1.5) in GBps
Intra link bandwidth rates	[500, 1000] KBps fixed value of Ethernet speed
Number of agents per site	1
Scheduling scenario	SOCIAL_NETWORK_GRID: is a Social Network Grid scenario, performing VO policy coordination with the help of referral agents recommendations.

5.5. 1 Scheduling Cost Functions

The scheduling of multi-component applications in a typical heterogeneous distributed environment, which aims at minimizing application completion time, requires a cost function to evaluate the potential set of candidate schedules. In this section, some of the main cost function that are of interest to us were analysed. These cost functions require information about the targeted application. For each user application component C_i, the following assumptions are made:

a. All inter-processor communications are contention free
b. In the cause of task execution, computation can be overlapped with communication.

The following cost functions are constructed using the information from user application and schedulable system resources, from a site S. Let N be the number of components, n the number of schedulable components, and m the number of sites,

If the earliest execution start time and earliest execution finish time (Topcuoglu *et al.*, 2002) of component C_i, on site $S_k$is represented by, $\Phi(C_i, S_k)$ and $\varphi(C_i, S_k)$ then,

$$\Phi(C_i, S_k) = max\{available[S_k]\ , max(\psi(C_i, S_k\ ) + comm(C_i, C_j, S_k, S_l))\} \qquad (5.6)$$

Where $\psi(C_i, S_k)$ is the actual finish time of component $C_i$in site S_k, available$[S_k]$ is the time when site $S_k$is free and ready to execute the component C_i. The nested inner max, returns the ready time when all the data needed by C_i has arrived at site $S_k$from the local site S_l.

The earlier execution finish time is given by

$$\varphi(C_i, S_k) = W_{i,k} + \Phi(C_i, S_k) \qquad (5.7)$$

where, $W_{i,k}$ is the time taken to execute component C_i in S_k

Then the computation time for component C_i running in site S_k is defined by a cost function given as:

$$comp(C_i, S_k) = W_{i,k} = \varphi(C_i, S_k) - \Phi(C_i, S_k), \qquad \forall i \leq n, and\ k \leq m \quad (5.8)$$

The average computation cost is given by

$$avg\_comp(C_i, S_k) = \overline{W}_{i,k} = \sum_{k=1}^{m} W_{i,k}/m, \ \ \forall i \leq n, and\ k \leq m \quad (5.9)$$

Where, $W_{i,k}$ is the estimated computation time to complete the execution of the component C_i running in site S_k, given the current state of the resources S_k is willing to provide to the application, assuming no other component is scheduled on S_k. And if $S_k$has some applications scheduled to run on it, then the computation function for the components may be degraded to reflect the sharing of $S_k's$ resources.

The communication time spent by component $C_i$in communication with C_j, when C_i and C_j are executed in site S_k and S_l respectively, is defined by the cost function given as

$$comm(C_i, C_j, S_k, S_l) = \rho_k + \delta_{k,l} \times \mu_{i,j}, \qquad \forall i, j \leq N, and\ k, l \leq m \quad (5.10)$$

Where, $\rho_k$is the communication startup time of S_k, $\mu_{i,j}$ is the amount of data transferred from C_i to C_j, and $\delta_{k,l}$ is the communication time per transferred unit from S_k to S_l.

The average communication cost of sending data from C_i to C_j is defined by

$$avg\_comm(C_i, C_j, S_k, S_l) = \bar{\rho} + \bar{\delta} \times \mu_{i,j}, \qquad \forall i, j \leq N, and\ k, l \leq m \quad (5.11)$$

Where, $\bar{\rho}$ is the average communication startup cost over all machines in S_k. $\bar{\delta}$ is the average communication cost per transferred unit over all machines in S_k. If there is no communication, then this value is 0. And if multiple components are sharing a link, then the $comm$ cost function for the components may degrade to reflect the sharing of the link.

The waiting time of component C_i, is the time C_i arrives at site $S_k$to the current time S_k is not able to process C_i.

$$waiting\_time\ (C_i, S_k) = CT_{i,k} - AT_{i,k} \quad (5.12)$$

Where, $CT_{i,k}$ is the current time taken by C_i in S_k without being processed, $AT_{i,k}$ is the arrival time of C_i into S_k.

5.5.2 Searching Algorithms Comparison

Comparatively to the Centralized Index Search (CIS) and Flooding Search (FS) algorithms, the proposed neural network based Index Search (NNIS) algorithm has proven to outperform the other two search methods, as shown in Fig. 5.5, Fig. 5.6, and Fig. 5.7 below. This improvement is as a result of the branching factor of the algorithm, which increases the parallel power of the search, by iteratively distributing the job query to the agent's neighbours in parallel, when the desired result is not obtained. As mentioned earlier, in the distributed index search strategy, each agent holds a copy of the index which often optimizes the probability of finding much faster the requested information about site capability, by keeping tracks of the availability of such computed data by each neighbouring site's agents.

The search comparison is based on the consideration of the elapse time, from the point of job submission to the point of successful reception acknowledgement, by the sender or requester. Different job submissions were made between the range of 10, 15 and 20. As shown in the Fig. 5.5, Fig. 5.6, and Fig. 5.7, the NNIS algorithm is the fastest to reach an almost complete (90%) semantic overlay network, while the FS and the CIS algorithms have a very similar behaviour. However, it was observed that the CIS seems to lose some performance to the other two algorithms towards the total completeness of

the network, as a result of the waiting factor induced by the centralized index search strategy (Fig. 5.7).

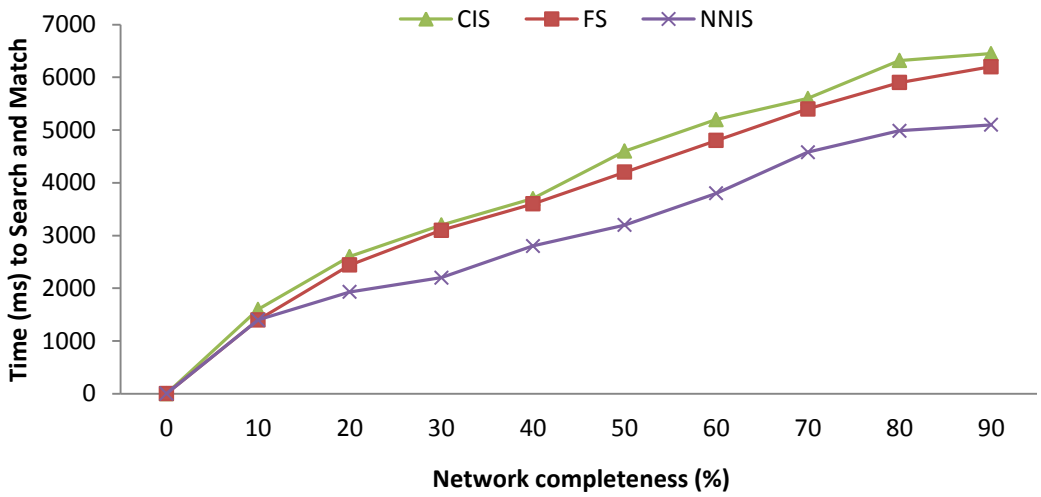


Fig. 5.5: Network Completeness Comparison Between FS, CIS, and NNIS when Considering 10 Job Submissions by a User

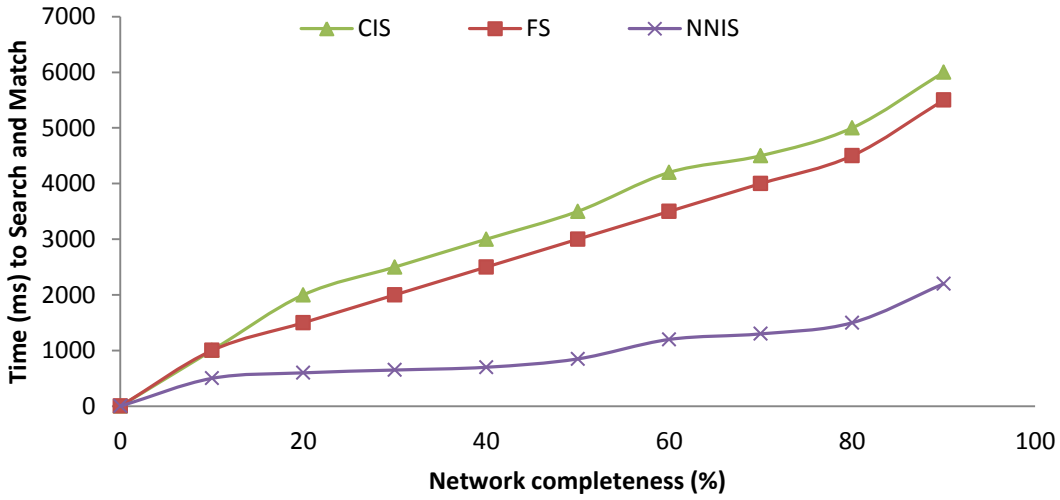


Fig. 5.6: Network Completeness Comparison Between FS, CIS, and NNIS when Considering 15 Job Submissions by a User

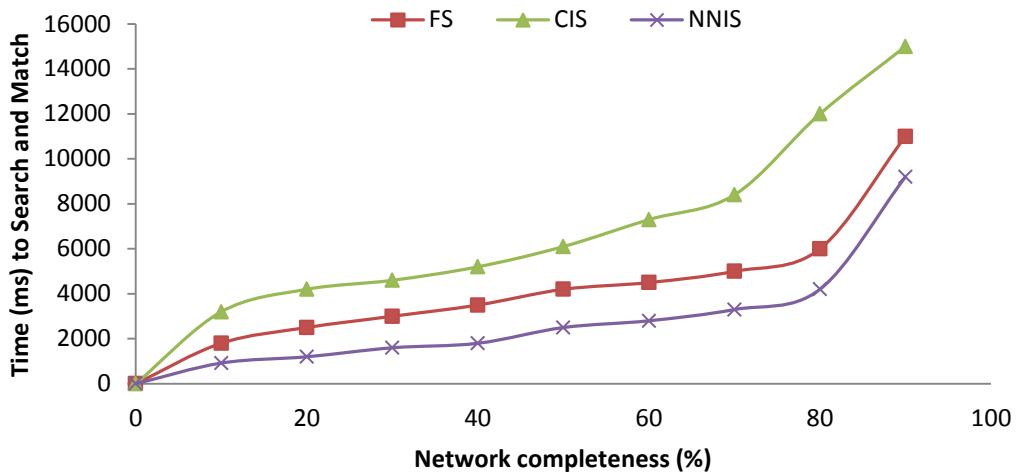


Fig. 5.7: Network Completeness Comparison Between FS, CIS, and NNIS when Considering 20 job Submissions by a User

5.5.3 Multi-component Scheduler Evaluation

Once the best resource has been discovered and selected, the scheduler generates a schedule for the job-resource mapping. This schedule consists of job characteristics, resource characteristics and job-resource mapping. The efficiency of any scheduling algorithm is measured by the time taken by the scheduler to generate schedules. So a graph is plotted between the number of components in multi-component jobs and the average running time of the scheduling process (or average time taken to generate a specific schedule). Fig. 5.8 depicts that, when the number of components in a job increases, the schedule generation time likewise increases in the initial stages, afterward, the algorithm running time is not affected by the increasing number of job components. The performance evaluation based on the various simulation executions, with respect to the increasing number of components in the multi-component jobs is shown in Fig. 5.8.

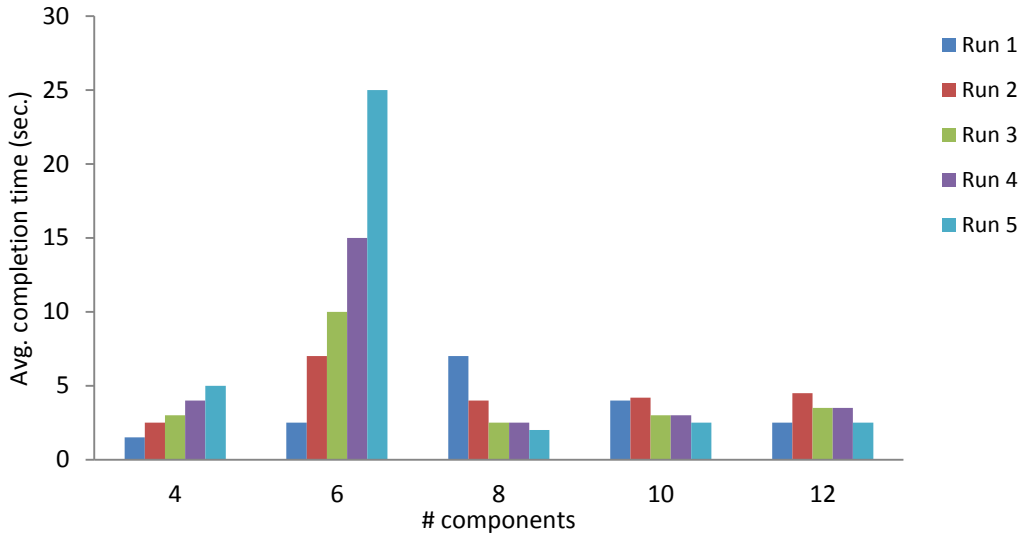


Fig. 5.8: Performance of Multi-Component Scheduling Algorithm

The graph shown in Fig.5.9 compares the average waiting time of the multi-component application before being executed by the selected available execution site. The proposed agent based resource selection strategy which is integrated with RI and denoted by *RI with Agents*, outperforms the remaining state of the art resource selection alternatives. As mentioned earlier on, the waiting time of the component C, is described as the time C arrives at site S to the current time site S is not able to process the component. This is computed for each arriving component as the difference in the component release time and the current time the component remains unprocessed. Next in performance is the least loaded, followed by the economic group based resource selection strategy, while the random based resource selection strategy seems to be the least of the three alternatives as shown in Fig 5.9.

The result of the experiment shown in Fig. 5.10 compares the performance of three resource selection strategies, including the *group economic-based* selection strategies, the *random* selection strategy, and the *least loaded* selection strategies, which achieves optimal scheduling (supposing perfect updated information on resources states) with the

proposed *RI with Agents* resource selection strategy. The resource load is calculated for each resource as the total queue length divided by the resource capacity. The *RI with Agents* resource selection strategy outperforms all the alternatives, scoring the closest to the *RI with Agents* selection method is the ideal least loaded selection strategy, followed by the Economic group selection strategy, with random selection strategy scoring the least.

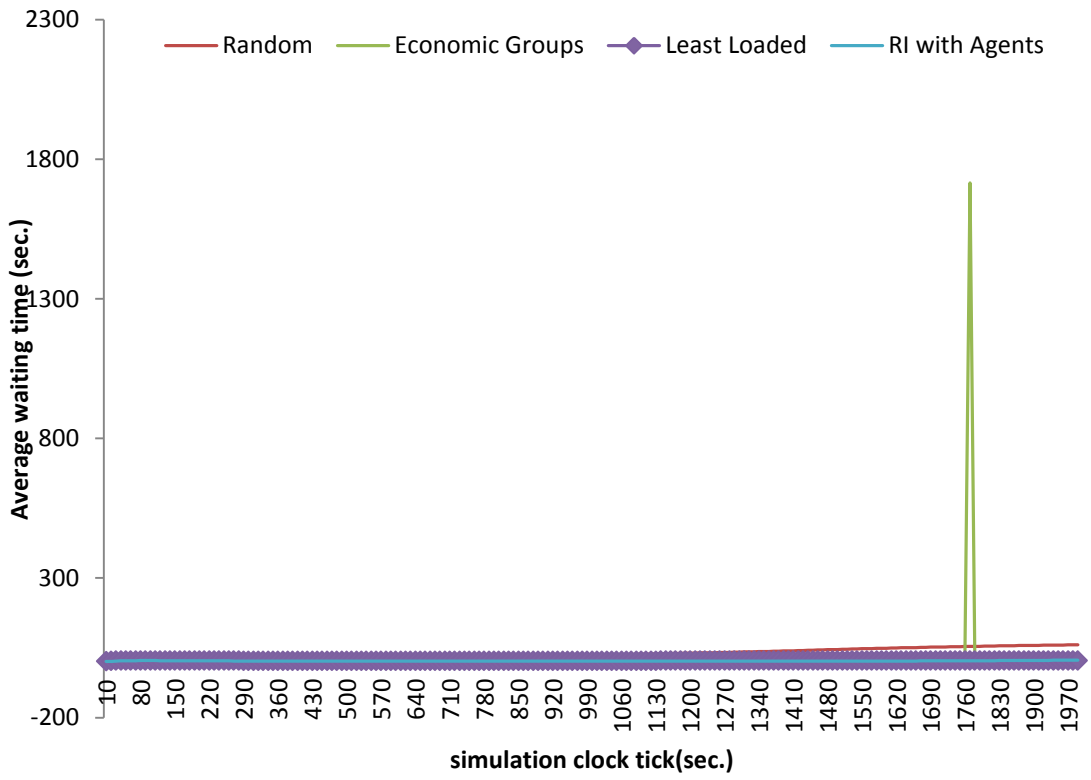


Fig. 5.9: Four Resource Selection Mechanisms Compared for Component Average Waiting Time

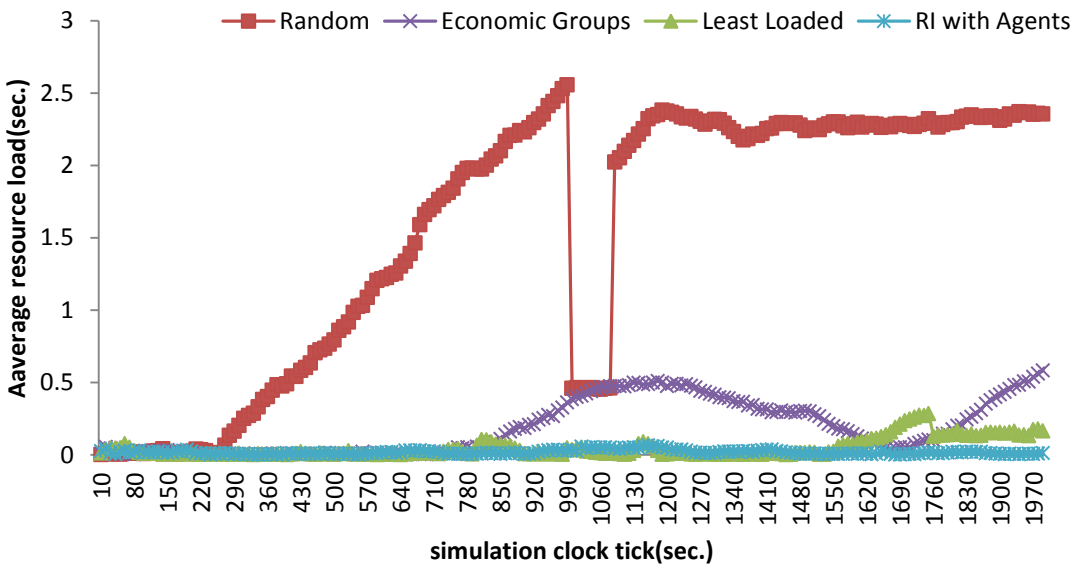


Fig. 5.10: Four Resource Selection Mechanisms Compared for Average Resource Load

The evaluation test cases involved using resources from different sites, connected to each other via active wide area network links. The proposed algorithm adapted well to the framework infrastructure, generating a distributed, dynamic and scalable algorithm, as each of the site contributes just one agent to finding solution to the overall scheduling problem. This helps in reducing congestion in communication routes, because tasks transfer only occurs between negotiating agents per site and not through intermediary.

More precisely, the proposed algorithm is based on the integration of Routing Indices (RI) with intelligent agents. By incorporating agents into the system, global scheduling logic is pushed to corporative agents that carry out coordination of job query forwarding, to neighbours that are more likely to have the desired computing resources. Usually, each execution site is assigned an agent coordinator, whose responsibility is to look for the suitable resource that matches the user job queries, by computing that site's resource capability. If an agent cannot find a suitable computing resource for a user job within its local site, it forwards the job query to a subset of its neighbours based on its

local HRI, rather than selecting neighbours at random or flooding the network by forwarding the query to all neighbours.

The results obtained from the simulation experiment carried out, show that the proposed resource selection strategy outperformed all the alternatives including; group economic-based, random, and least loaded resource selection strategies. Scoring the closest to the *RI with Agents* selection method is the ideal least loaded selection strategy, followed by the Economic group selection strategy, with random selection strategy scoring the least.

5.6 Evaluation of Proposed Matchmaking Algorithm (Experiment 3)

A simulation based test is adopted to verify the efficiency of the proposed matchmaking algorithm. The Alea Grid simulator (Klusácek and Rudová, 2010) with an extension of Agent Repast and GridSim (Sulistio, 2008) was used to compare the performance of the agent-based matchmaking algorithm against other scheduling algorithms. Ideally, Alea with an extension of Gridsim provide a comprehensive facility for simulation of different classes of heterogeneous resources, users, applications, resource brokers, and schedulers. The simulator serves as good benchmark which can be used to simulate application scheduler for single and multiple administrative domain(s) computing resource environment. Alea has support for visualization tools suitable for graphical display of scheduling performance results. However, it does not provide support for agent based scheduling and as such, the generic agent-based Grid simulator specifically built for developing agent coordination mechanism on top of Grids (Isaac, 2014) was extended and used for the evaluation of the proposed work.

5.6.1 Local Scheduling Policies

The sole aim of the Matchmaking process is to find a more suitable resource node to allocate the user job request, after which the execution process start up at the remote

local machines that was selected by the agents. It is however, necessary to carry out the tasks of job execution most efficiently, so as to comply with the delivery time requested by the users. Normally, since the job execution process is online and dynamic in nature, the job estimation and assignment processes, must be performed during the execution time. Information pertaining to the current system state plays a vital role in the process of job allocation. For instance, information about all the jobs existing in the waiting queue and the current state of the node must be considered. In this section, some of the key local scheduling metrics that can be used to determine the efficiency of the proposed scheduling system are given as follows.

a. Job Waiting Time: the period of job arrival to the current time that the job stayed without being processed. This variable is updated dynamically over time:

$$Job\_Waiting\_Time = ET - ST \tag{5.13}$$

Where ET and ST are respectively the job execution start time and job submission time.

b. Average Waiting Time: this is given by

$$Average\_Waiting\_Time = \frac{\sum_{i=1}^{C}\left(ET_i - ST_i\right)}{C} \tag{5.14}$$

Where C represent the maximum number of job.

c. System Utilization: the average resource utilization rate is given by

$$System\_Utilization = \frac{\sum_{j=1}^{K} SUR_j}{K} \tag{5.15}$$

Where SUR denotes the system utilization rate of resource belonging to site j and K denote the number of sites

d. Communication cost: In heterogeneous computing environment, each platform is usually connected with different bandwidth sizes. So, if components take the same time to communicate between them, then they are allocated to better bandwidth

links and it delivers more effective results. However, this is often not the case, as there are variations in the bandwidth sizes and that can limit parallel execution of interacting components. Therefore, the communication cost of any specific running jobs with interacting tasks is necessary. The communication cost function for a component is given by (Weissman, 2000),

$comm\,[C_i,\ C_j,\ S_k,\ S_l]\ \forall_{i,j,k,l}(i,\ j < N),\ (k,\ l < m)$. Where, S is a site, N is the number of components, n is the number of schedulable components, and m is the number of sites. This function gives the communication time spent by component C_i in communication with component C_j when C_i and C_j are run in site S_k and site S_l respectively.

e. Total Communication cost: The total communication cost for a component is a function of the communication cost values associated with its links. This is expressed as follows:

$$[C_i,\ S_k] = \frac{TotalCommSize_i}{bw[i,k]}, \forall_{i,k}(i \le n),\ (k \le m) \qquad (5.16)$$

where $TotalCommSize$ is total communication size of component C_i and $bw[i,k]$ is the bandwidth between site i and k.

f. The completion time for the scheduling of multi-component application can be defined in terms of the component cost functions (Weissman, 2000).

$$CT_{serial\_flow}[C_i, C_j S_k, S_l] = \sum comp[C_i, S_k] + total\_comm[C_i, C_j, S_k, S_l] \forall (i,\ j \le N), (k, l \le m) \qquad (5.17)$$

$$CT_{networked\_flow}[C_i, C_j S_k, S_l] = \max\{comp[C_i, S_k] + total\_comm[C_i, C_j, S_k, S_l]\} \forall (i,\ j \le N), (k, l \le m) \qquad (5.18)$$

Where, the function $comp$ gives the computation time for the component C_i running in site S_k give the current state of the resources S_k is willing to provide to the application

assuming no other component is scheduled there. If other components are also scheduled in S_k, then the $comp$ function for the components may be degraded to reflect the sharing of $S_k's$ resource.

In this section, the effects of one other related job criteria, that is suitable for measuring the performance of the proposed matchmaking algorithm is considered. This criteria is the $slowdown$. Slowdown in this case, is the ratio of the actual response time of the job to the response time if executed by the system without any waiting. By definition, according to (Klusáček, 2008), $Slow\_Down > 1$. However, the use of response time places more weight on long jobs and basically ignores if a short job waits few minutes, so it may not reflect users' notion of responsiveness. Slowdown reflects this situation, by measuring the responsiveness of the system with respect to the job length, in which case, jobs are completed within the time proportional to the job length. The $Slow\_Down$ criteria is expressed as follow,

$$Slow\_Down = \frac{1}{|NFJ|} \sum_{j \in NFJ} \left(\frac{response\_time}{run\_time} \right) \quad (5.19)$$

Where, NFJ is number of non-failed jobs. The $response\_time$ represents the average time a job spends in the system, i.e., the time from its submission to its termination. While run_time denotes the completion time of the last executed job.

Klusáček (2008) explained that extremely short job having very small runtime ($e.g., 0 < run\_time < 1$) is dangerous. As it often represents job that ended prematurely, right after the start due to some error. However, its slowdown can be huge, which may seriously skew the final mean value. Therefore, the $Bounded\_Slow\_Down$ is often applied (Dror, *et al.*, 1997 and Dror, 2005), where the minimal job runtime is guaranteed to be greater than some predefined time constant, sometimes called a "threshold of interactivity" (Dror, *et al.*, 1997).

5.6.2 Experiment Setup

The experiment involved scheduling of components of problem size 5000, 103656 for the first test case, and 59700, 202876, 96000, 121038, 103656, 187370, 42724,96000, 121038, 103656, 187370, 42724, for the second test case that were originally executed, in the first 18 days on 14 clusters having 806 CPUs, and processors, with CPU speed of 1.25-2.50GHz. The job load and the configuration of the clusters' characteristics used for the simulation test are contained in two separate files downloaded from an open domain, provided by the Czech National Grid Infrastructure (MetaCentrum, 2014). Through the experiment, three different scheduling algorithms were compared: The selected algorithms are FCFS (First Come First Served), ESG-LS (Earliest Suitable Gap - Local Search), and MA-SP (Multi-agent - Scheduling Policy). MetaCentrum is composed of 14 Linux clusters, with different configurations, as shown in table 5.12:

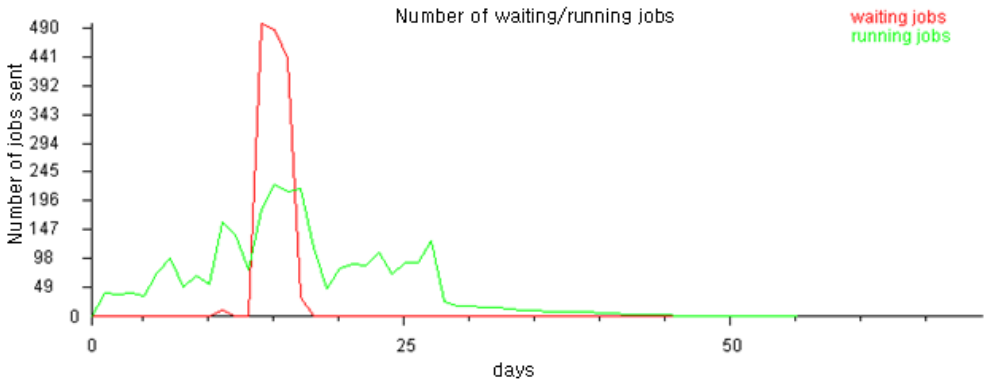
Table 5.12: Number of Processors and Nodes in the MetaCentrum

Cluster	Processor	Speed	Nodes	Total CPUs
cluster_0	Itanium	2 .5GHz	8	8
cluster_1	Opteron	2.2GHz	16	16
cluster_2	Xeon	3.2GHz	10	10
cluster_3	Opteron	2.6GHz	5	80
cluster_4	AthlonMP	1.6GHz	16	32
cluster_5	Xeon	2.4GHz	32	64
cluster_6	Xeon	2.7GHz	36	148
cluster_7	Xeon	3.1GHz	35	70
cluster_8	Opteron	1.6GHz	10	20
cluster_9	Opteron	2.4GHz	3	6
cluster_10	Opteron	2.0GHz	23	92
cluster_11	Xeon	3.0GHz	19	152
cluster_12	Xeon	2.7GHz	8	64
cluster_13	Xeon	2.3GHz	11	44

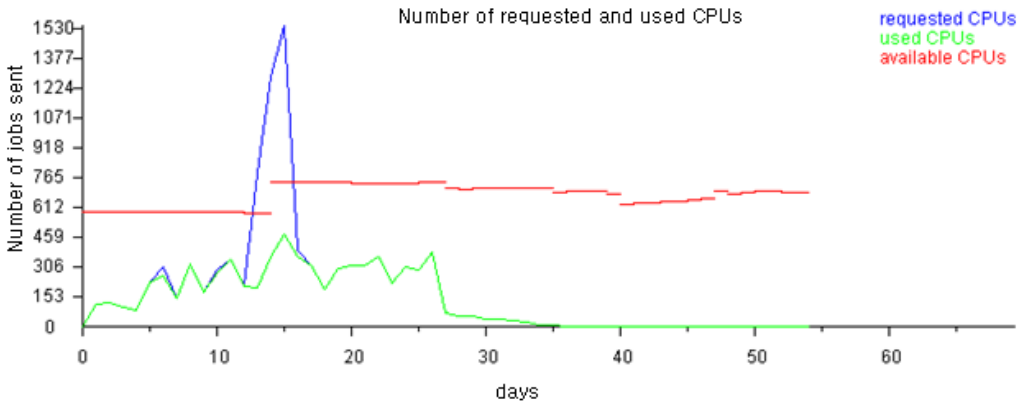
The graphs in Fig. 5.11 and Fig. 5.12 demonstrate major differences between the two algorithms and a scheduling policy compared for their performances. In the first criteria, the number of running jobs was compared against the number of waiting jobs resulting from the ESG-LS and MA-SP. From Fig. 5.11, even though the ESG-LS is a good

scheduled based heuristic algorithm, that is capable of a higher resource utilization and reduction of the number of waiting jobs through the time, it still generates reasonable peak of waiting jobs during the execution time. The MA-SP produces the best results with an indication of optimal resource utilization (Fig. 5.12a) and increased number of running jobs as indicated by the high multiple peaks of running jobs in the course of application run (Fig. 5.12b).

The second test case scenario, compares the FCFS scheduling algorithm and the MA-SP as illustrated in Fig. 5.13 and Fig. 5.14. Concerning the machine usage (Fig. 5.13a) as expected, FCFS generates very poor results. FCFS is not able to utilize available resources when the first job in the queue requires some specific and currently unavailable machine(s). However, the policy-based MA does not apply such restrictions and thanks to the application of a more efficient policy-based approach, it produces the best results (Fig. 5.14a and Fig. 5.14b).



a.)

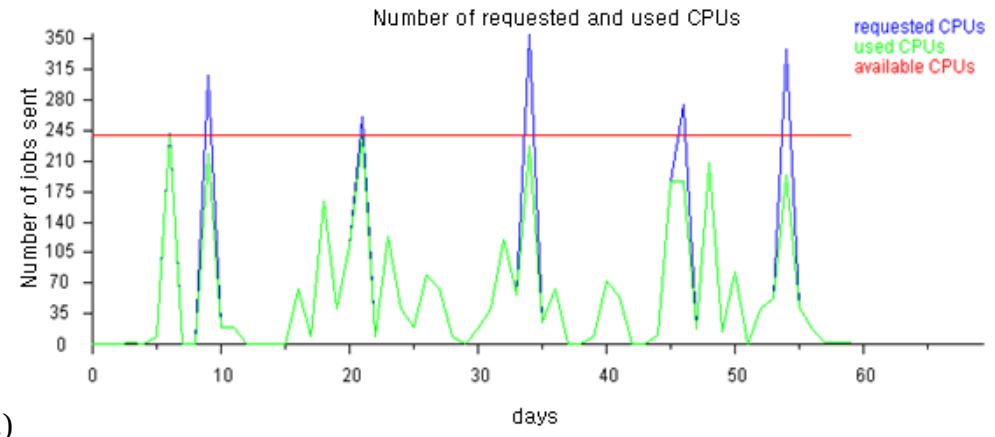


b.)

Fig. 5.11: Scheduling of Components with Problem Size = 5000 Using ESG-LS

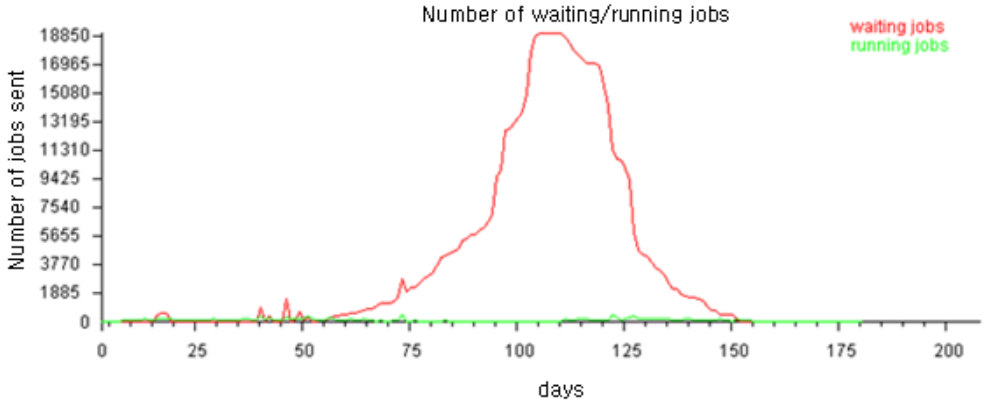
The peculiarity in the behaviour of the plot in Fig. 5.1b, with regards to available CPUs, is due to the dynamic nature of the computing environment. This dynamic behaviour can sometimes translate into some level of inconsistencies in the availability of both the network and computational resources (as is the case with the line break in Fig. 5.12).

a.)

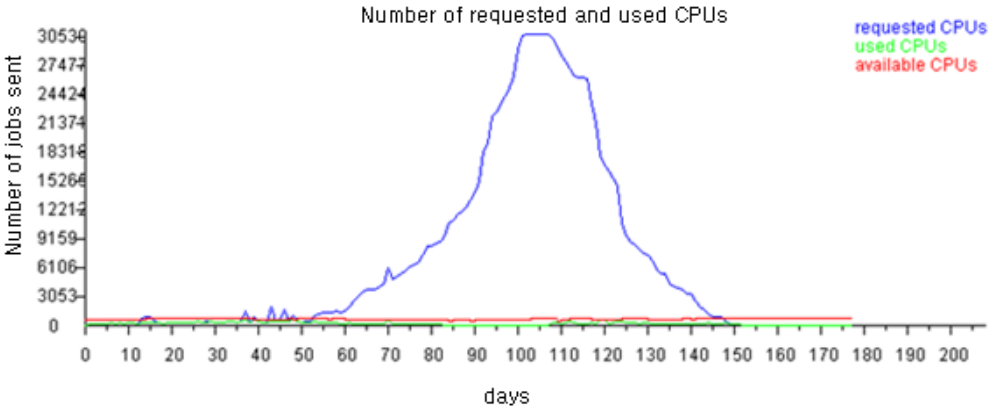


b.)

Fig. 5.12: Scheduling of Components with Problem Size = 5000 Using MA-SP



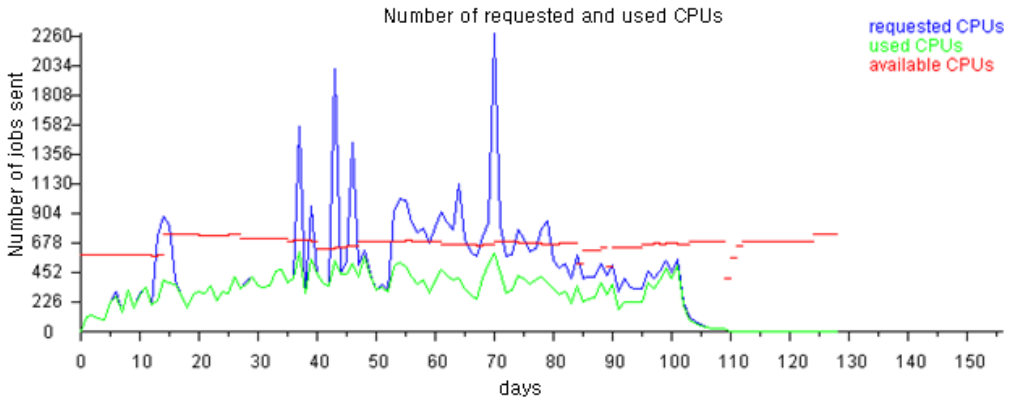
a.)



b.)

Fig. 5.13: Scheduling of Components with Problem Size = 59700 Using FCFS

a.)



b.)

Fig. 5.14: Scheduling of components with problem size = 59700 using MA-SP.

Another simulation experiment was carried out by using the same simulator to determine the optimal percentage completion time for the two types of application flow discussed in this dissertation. The two scheduling policies were tested over a large set of simulated multi-component platforms and applications, and measured their performances with respect to the optimal schedule generated. The multi-component resources and applications data used for the simulation, were obtained from the work done in (Weissman, 2000) and presented in table 5.13. The multi-component platform comprises of, networks of computing sites, each with varying associated communication bandwidths, while, the multi-component applications consists of

compute-intensive and data-intensive components, each with varying resource requirements capabilities.

Table 5.13: Simulation Parameters

Parameter name	Value range
Number of components	2, 4, ... 12
Number of sites	1, 2, ..., 15: [S1, S2 ... S15]
Inter link bandwidth rate	[0.5, 2] GBps
Inter link bandwidth rates between sites	S1-S2:(1), S1-S3: (1.5), S2-S4:(1.2), S2-S5:(1.5), S3-S6:(1.5), S3-S7:(1.5), S4-S8:(0.9), S4-S9:(1.5), S5-S10:(0.9), S5-S11:(1.5), S6-S12:(0.9), S6-S13:(1.5), S7-S14:(0.9), S7-S15:(1.5) in GBps
Intra link bandwidth rates	[500, 1000] KBps fixed value of Ethernet speed
Number of agents per site	1:(total number of agents for the 15 sites is 15)
Scheduling scenario	SOCIAL_NETWORK_GRID: is a Social Network Grid scenario, performing VO policy coordination with the help of referral agents recommendations.

Generally, the scheduling policy performed well for the two types of application flow considered in this dissertation (Fig. 5.15 and 5.16). This hierarchy of optimal performance shows the level of increasing sensitivity of the overall optimal completion time, to the scheduling of a single component application. The result depicted in Fig. 5.16 can be attributed to the fact that, for the serial flow application, which is scheduled using the Improved Close-to-File (ICF) algorithm, the completion time, is the sum of all interacting components. Therefore, the policy can afford to schedule a few components sub-optimally. For the networked flow application type, which is scheduled with the First-Fit (AFFRSP) algorithm, sub-optimal scheduling of a single application can have a significant impact on the completion time, as applications (with interactive tasks) are executed on more than one node. However, the two algorithms performed quite well,

when the number of component is = 6, both policies are within 8% of optimal on average. In general, they have performed reasonably well in both cases of application flow, within an average of 20% optimal completion time for components less or equal 12 in number.

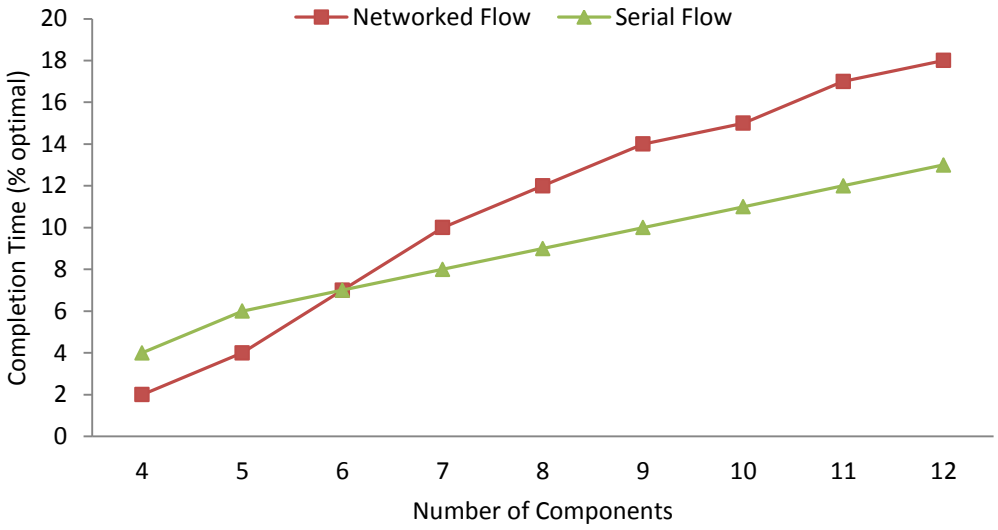


Fig. 5.15: Scheduling Policy Performance with Increasing Number of Components.

Result from Fig. 5.16 shows that the First-Fit scheduling policy is sensitive to the number of execution site involved in the execution of the application components. This is obvious as the execution performance showed some level of improvement for a while and afterwards starts to degrade slowly as the number of active site increases. The ICF appeared to be insensitive to the increasing number of execution sites. This is not surprising, since the ICF targets the scheduling of application on a suitable single cluster.

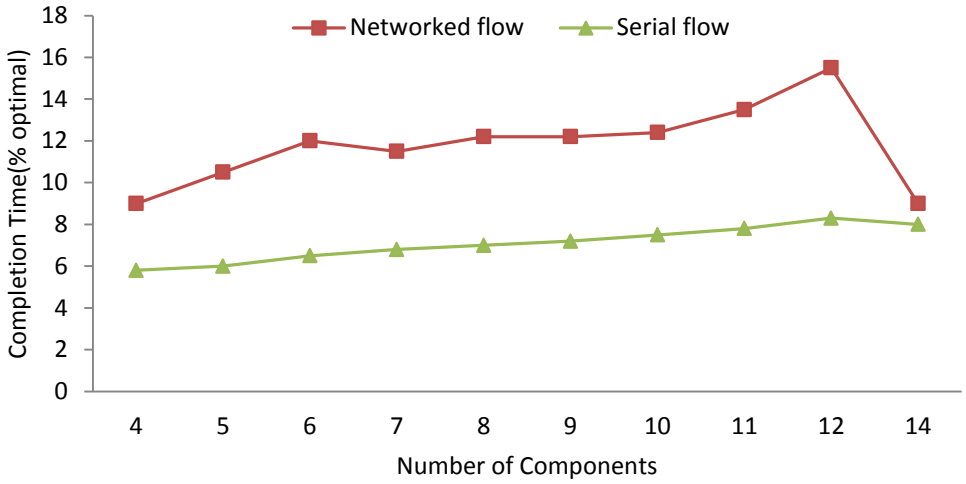


Fig. 5.16: Scheduling Policy Performance with Increasing Number of Sites

5.6.3 Performance Evaluation of the Matchmaker Algorithm

The Alea grid simulation environment implements several algorithms, that were executed in all our test cases, introduced to evaluate the performance of the proposed scheduling solution. However, it was discovered that some of the algorithm used to compare the relative performance of the proposed work had stopped functioning in the course of scheduling process, due to lack of sufficient information on the job load, resource characteristics and other relevant information, that would have guided the scheduler to efficiently make scheduling decisions.

A log file of the job load generated by the resource manager of the Alea grid simulator is used to identify 143 system users, that send 59700 jobs to the grid over a period of 300 days. Fig. 5.18 shows the jobs per day that arrive at the grid in 300 days. Four job arrival peaks can be observed that are outside the average job load, as it is uncommon for the grid to exceed 2000 tasks per day. These differences in the rates of task arrival serve to evaluate different aspects of the algorithm with a job load extracted from a real environment.

The first peak of Figure 5.17, which occurs specifically on days 62, 63, and 64, is chosen for the purpose of analyzing the behaviour of the algorithm with non-uniform job arrival rates. Varying number of jobs sizes are sent to the different selected nodes at different time, and their total processing time estimated, the peak shows that the jobs enters in a more balanced manner despite the un-normalized entering rate of the jobs across all the nodes. The largest task peak appears on days 179 and 180, when 59700 and 103656 jobs arrive, respectively. This number of tasks is outside the normal arrival range of jobs. However, the jobs also enter the system in an unbalanced form.

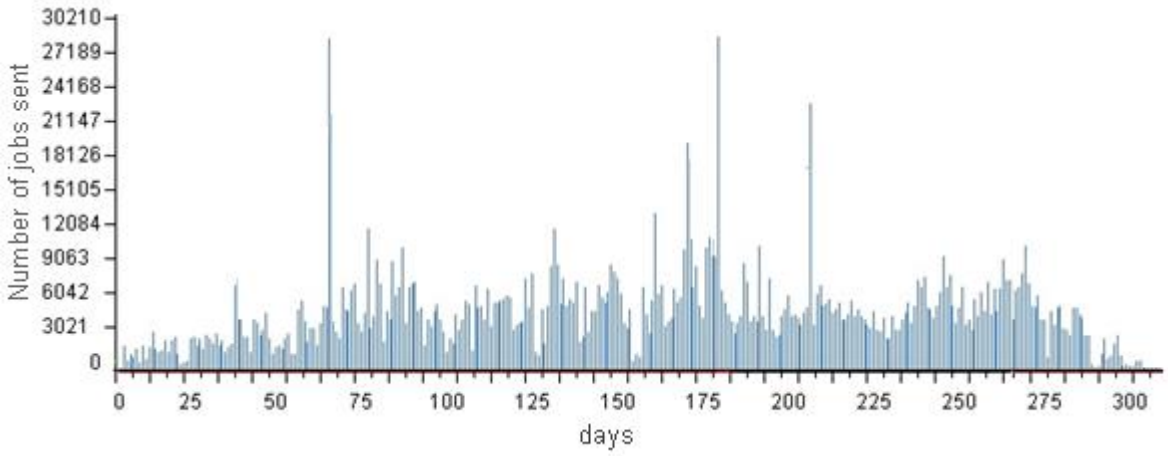


Fig. 5.17: Number of Tasks Per Day that Enter the Grid.

Fig. 5.18 below, illustrates the average machine usage per day. It demonstrates that the load at each machine behaves in a balanced way at the time of job execution. The observed trend in the machine performance characteristic, could be attributed to the careful selection of most suited machines for every user jobs on queue. It was observed that, the assignment of job by the scheduler respects the time windows requested by users, who send jobs for execution, since most of the submitted jobs are usually processed early enough even before their due dates are reached.

The trend in the machine load would appear to be consistent throughout the execution process. The reason being that for every job submitted into the system, there are commensurate resources selected for each job execution. Therefore, the initial intelligent selection of resource is paying off at this time. Again, the arrival rate of jobs does not directly affect the occupation of each machine or its load, since, when the arrival rate increases; the jobs tend to be executed in the machines with largest capacity. This phenomenon is depicted in Figure 5.18. The event of assigning jobs to machines with larger execution capacity is more likely to occur, because assigning a job to that machine would have a lower impact on the overall machine load, making the scheduling agents more likely to select it, because from the beginning of the increase in the arrival rate of jobs up to the time the receiving agents would updates its local state, the other agents continue to try to negotiate the execution of jobs with the machine.

In the proposed framework, agents are attached to each machine or nodes. Therefore, machines are not usually engaged 100% fully, even with the continuous increase in the arrival of jobs into the system, because there are time slots in which agents are supposed to start executing previously negotiated jobs on queue and at the same time try to update their local states information into the object base storage, for other resource agents to access and know about their current status. The agents at the resource level would usually, have the same intent of collaborating with each other to reduce job execution time, while maintaining load balance on each machine.

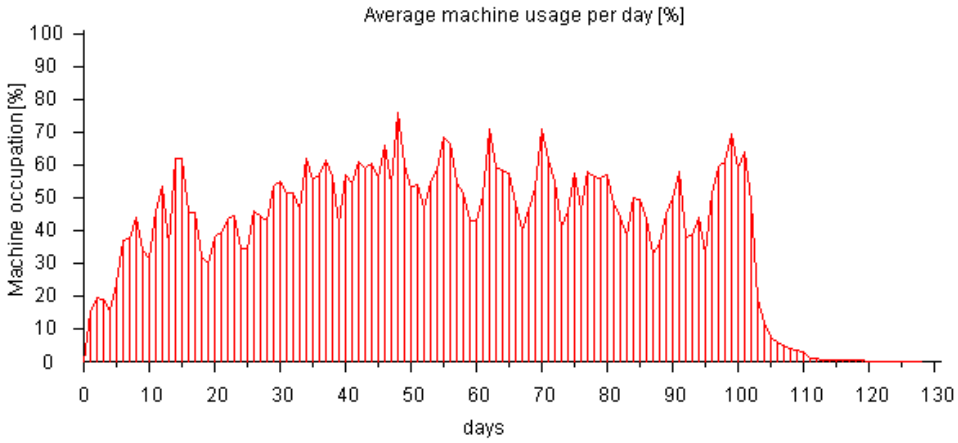


Fig. 5.18**:** Total System Utilization Using the Guaranteed Strategy on the High Workload Without Resource Reallocation.

In Fig. 5.19, some irregularities were observed in the peak pattern depicted in this Figure shown below. When comparing the results of the proposed algorithm with those of five other algorithms including; FCFS, EDF (Earlier Deadline First), (EASY Backfilling), AggressiveBF (Aggressive Backfilling), SJF (Shortest Job First), and FairShareFCFS. Most of these algorithms stopped functioning due to lack of sufficient information on the job load or job resource requirements (see the peak down in Fig. 5.19). This can be traced from day 65 to day 110. The process is started again, with the introduction of resource reallocation, for which, some time, is spent on searching for more suitable resources after every failure occurrences (Machine failure information is also part of the data set from the Grid infrastructure MetaCentrum). A high peak of close to 100% is noticed in day 155 and afterward a drastic drop is seen. This can be attributed to the inconsistency in the choice of resource selection. With resource reallocation, there are chances that low capacity resource can be assigned to a specific job, which can lead to the observed irregularity seen in Figure 5.19.

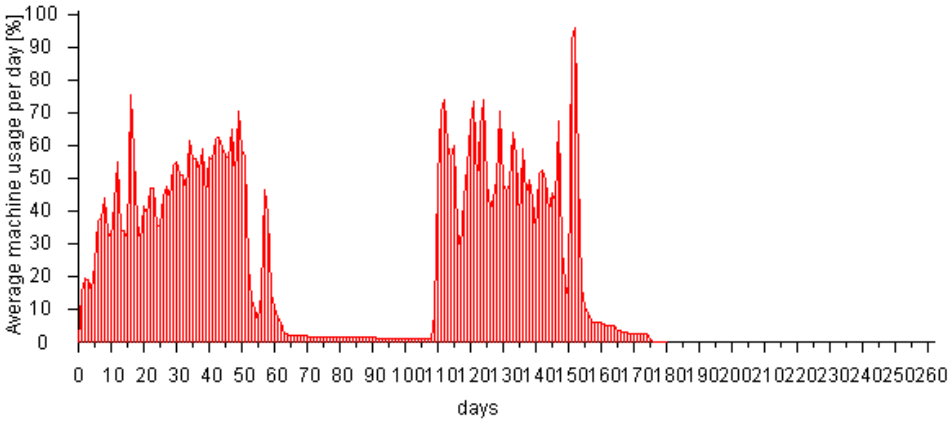


Fig. 5.19: Total System Utilization of the High Workload with Resource Reallocation

5.6.4 Performance Evaluation Based on Job Related Criteria

In this section, the effects of the $Bounded\_Slow\_Down$, discussed in section 5.5.4 was considered, this is one of the job related criteria that is suitable for measuring the performance of most job scheduling algorithms, and in this case, the proposed matchmaking algorithm presented in this dissertation. As discussed in Section 5.5.4, slowdown measures the responsiveness of the system with respect to the job length, meaning that jobs should be completed within the time proportional to their length. During the experiment, six well known algorithms were compared with the proposed MAS algorithm, including the FCFS, EASY, EDF, and a combination of Conservative Backfilling (CONS), Gap Search (GS) and Random Search (RS) algorithms.

The Fig. 5.20 shows the cumulative distribution functions (James, 2009) of bounded slowdowns for the MetaCentrum'09 experiment. The x-axis depicts the bounded job slowdown, while the y-axis describes the probability of a job falling within the expected range of responsiveness. The cumulative distribution functions are $f(x)$-like functions which shows the probability that a given bounded job's slowdown is less than or equal to x. Fig. 5.20 clearly demonstrates the general improvement obtained by MAS-

proposed algorithm with respect to the percentage of jobs having their bounded slowdown 1000. However, it does not indicate why the difference in the average value is so large. The reason lies in the cumulative distribution functions "long trail" which is displayed in Fig. 5.21. Clearly, the maximum bounded slowdown is always less than 1000 for MAS-proposed algorithm, while an estimated large number of jobs have their bounded slowdown bigger than 1405 when other algorithms are used. Apparently, the evaluation phase of the MAS-proposed algorithm is capable to identify such extremely bad decisions and fix them. Since the remaining algorithms with the exception of Cons+RandomSearch do not use evaluation they cannot identify nor fix these types of problems. However, the Cons+RandomSearch give a close match result to the proposed MAS algorithm with the runtime optimizing capability of the random search policy.

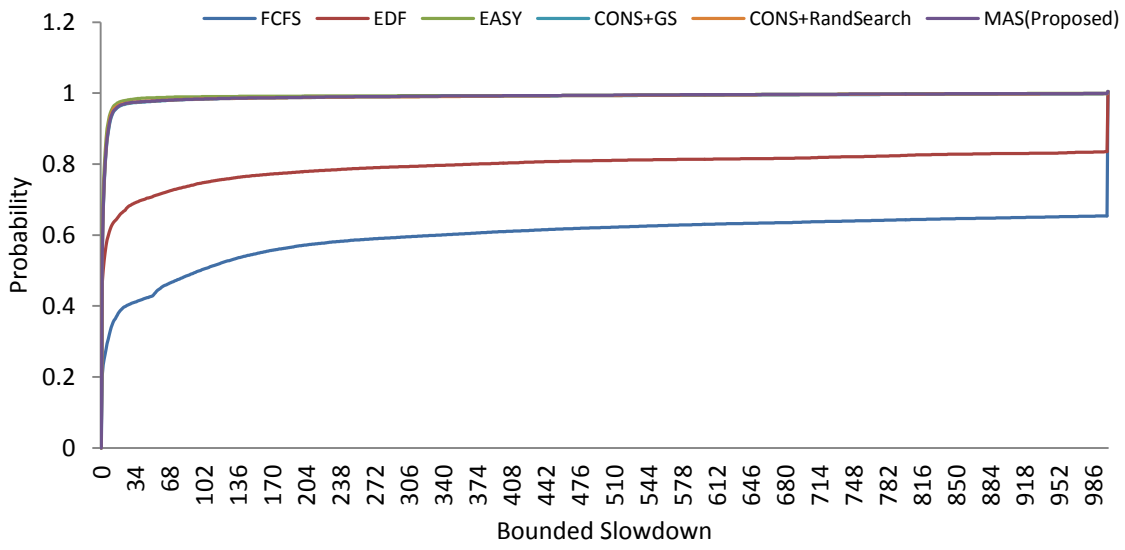


Fig. 5.20: The Cumulative Distribution Functions of Bounded Slowdowns in MetaCentrum'09 Bounded by 986

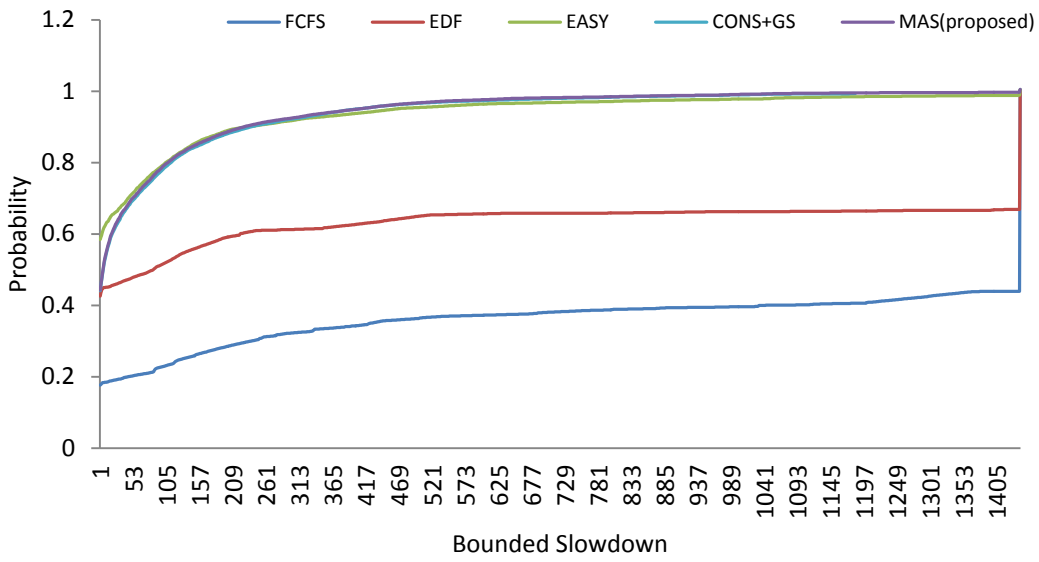


Fig. 5.21: The Cumulative Distribution Functions of Bounded Slowdowns in MetaCentrum'09 Bounded by 1405.

In another aspect of the experiment, the fairness of the proposed framework with respect to its user waiting time was determined. The user waiting time normally referred to as the normalized user waiting time is computed as the ratio of the total user waiting time to the total squashed area metrics proposed in (Carsten et al., 2004). The closer the user waiting time value, the higher the fairness of the system utilization of its resources. The implication of this demonstration shows that, when the commutated ratio of the normalized user waiting time < 1, the system user would have spent a significant amount of time computing than waiting. In other words, the system user computation time out-weighs its waiting time. The graph in Fig. 5.22 shows that the normalized user wait time is usually very low with respect to the utilized CPU time for the proposed scheduling solution. The normalized user waiting time is usually large for other algorithms due to their non-prioritized job queue lengths.

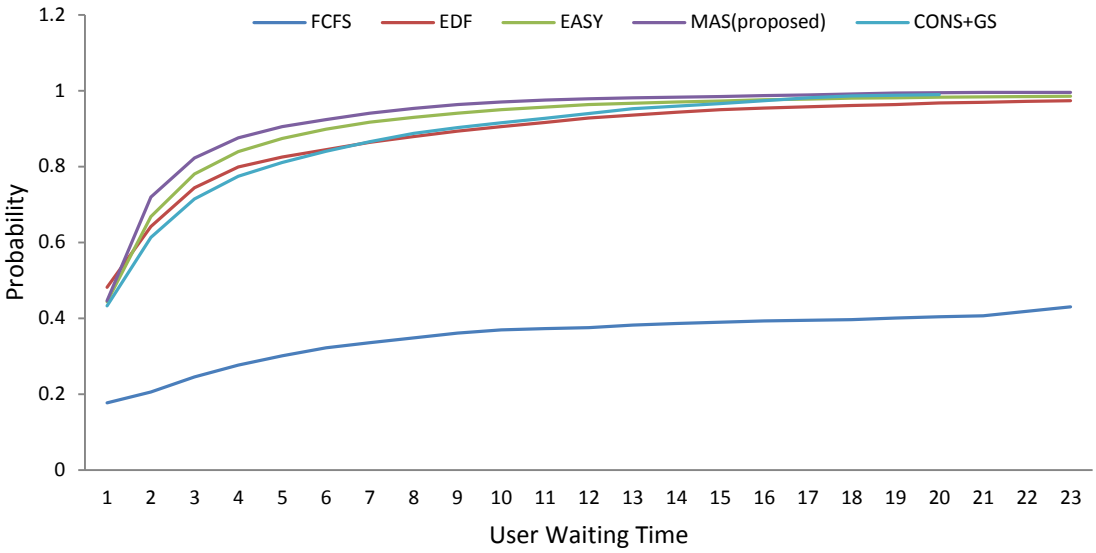


Fig. 5.22: The Cumulative Distribution Functions of User Waiting Time for MetaCentrum'09

A distributed scheduling simulation framework was extended to incorporate agents; tests of the proposed scheduling policies were performed on the framework. The results showed that scheduling with agents as an alternative approach to the heuristic-based scheduling algorithms methods, appeared to provide better scheduling results with respect to reduction in execution time, reduction in the number of waiting job as against number of running jobs and a remarkable increase in the number of used CPUs as against number of requested CPUs.

The obtained results suggest that effective and efficient scheduling of multi-component applications is possible, if sufficient application and system component information are supplied to the scheduler a priori, part of which include taking into cognisance the communication costs between components. However, the norm is usually to ignore the communication and confine application to run in a single site.

5.7 Evaluation of Task Scheduling With Agent and Without Agent (Experiment 4)

The proposed implementation was tested outside the conventional Grid simulator environment, by developing a simulation application in MATLAB to carry out the test experiments. For the MATLAB application, each simulation experiment ends when 10000 tasks executions gets completed. The arrival of tasks at time interval t is modeled as Poisson random process given by $P(t,\lambda) = \frac{e^{-\lambda}\lambda^{t}}{t!}$ (*5.20*)

Where, λ is the rate of arrival of the tasks to nodes.

To evaluate the performance of the two proposed scheduling policy, the following two types of tasks flow were considered; Data intensive tasks and Compute intensive tasks. Sixteen (16) nodes were considered for the computation. First, the scheduling policy without agent based scheduling was tested, where tasks are randomly assigned to any of the available nodes regardless of task type, but in agent based scheduling, first of all, incoming task type is determined and allocated to appropriate node i.e. data task is assigned to data specific node and compute intensive requirement tasks are assigned to compute intensive specific nodes since agent has knowledge of all resource in the system. Nodes are selected and matched based on task type. Proximity of job files to nodes is also considered, as is the case with close-to-file scheduling policy.

In the ANN-based agent intelligence scheduling, if a task arrives, it is directly forwarded to the execution node, where the same type of task has been executed previously, since agents have been trained to adapt to this type of scheduling scenario. In addition, the object base storage system keeps track of all information about previous tasks execution processes, this is usually updated by each group agents, involved in the scheduling events. The performance of the scheduling solutions without agent and with

agent based system was evaluated. Tasks are schedule using the two scheduling policies, i.e. First-Fit and Close-to-File Scheduling policies discussed in section (4.3.3 and 4.6.4), of this dissertation.

For each simulation run, the scheduling length SL or makspan is computed as follows:

$$SL = \max(EET(task_k)) \tag{5.21}$$

Where EET is the Estimated End Time of $task_k$ on $node_i$.

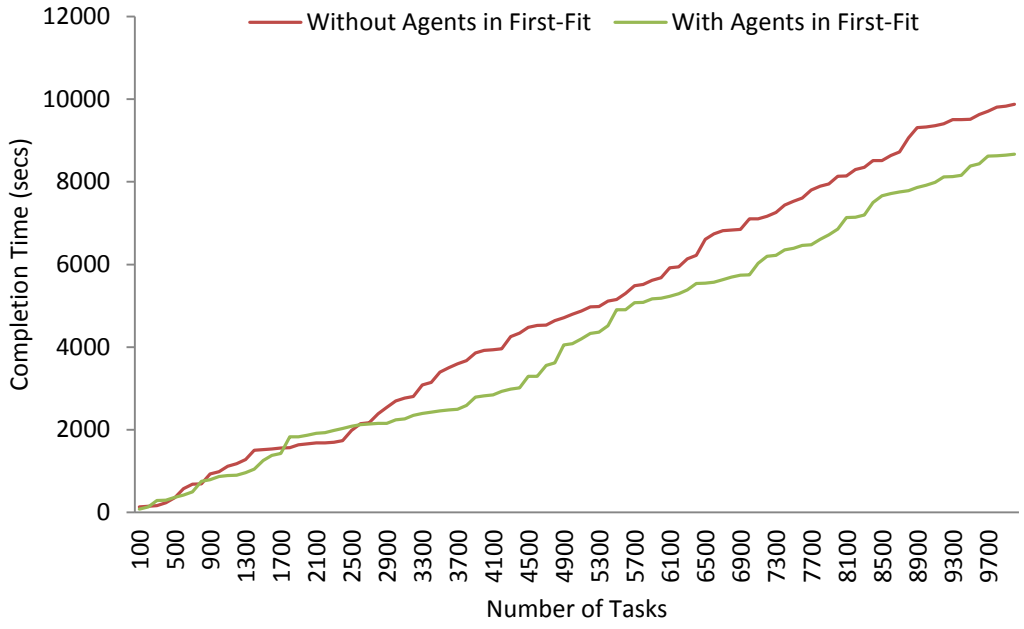


Fig. 5.23: Makespan Comparison using First-Fit Scheduling Policy

The performance of the First-Fit scheduling policy is due to the fact that, it targets applications, which resource requirements goes beyond what a single cluster can offer (multi-component applications), and as such, multiple clusters are requisite for the execution of these classes of applications. This policy is also efficient in executing networked applications with multiple interactive tasks. However, in the case of the Improved Close-to-File scheduling policy, it is efficient in terms of scheduling single-component applications.

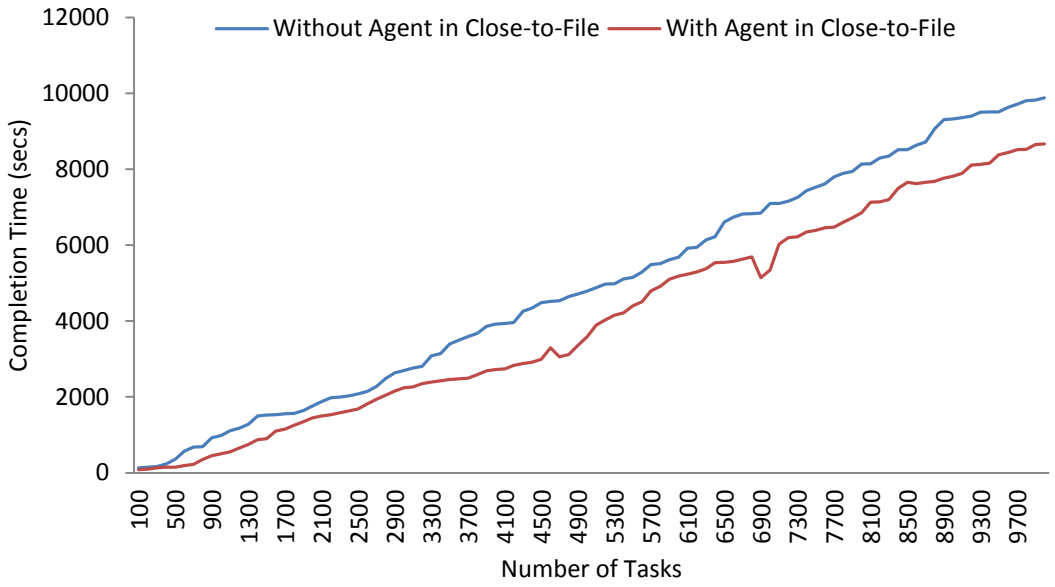


Fig. 5.24: Makespan Comparison using Improved Close-to-File

The simulation results are as shown in Fig. 5.23 and Fig. 5.24. These results clearly indicate that there is a remarkable reduction in the scheduling length with agent based scheduling, as compared to the scheduling without agent. The performance of the non-agent based scheduling turns out to be poor as compared to agent based task scheduling strategy. This can be attributed to the high level of intelligence, coordination and collaboration exhibit by the agents.

5.8 Performance Evaluation of Agents Network Traffic Load (Experiment 5)

The simulation parameters and the performance indices used in our analysis were configured as follows: The number of grid sites/nodes was varied from 10 to 50. One or more agents are generated for each site with a given probability. However, it was observed that, a higher cost of agents' activity is incurred with an increase in the number of agents, which also causes an increase in the traffic load (Fig. 5.25). A correct setting of the value of agents should take into account the trend of these performance indices

and in general should depend on system features and requirements, for example on the system capacity of sustaining a high traffic load.

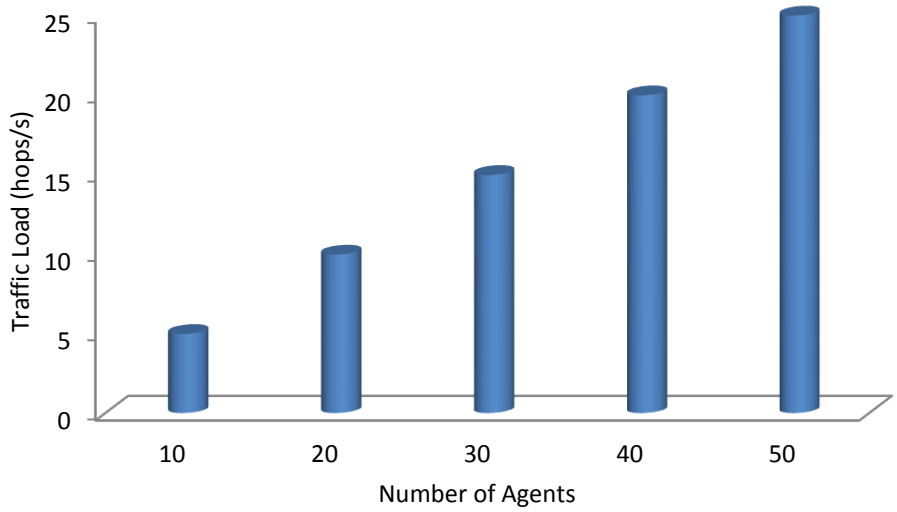


Fig.5.25: Mean Network Traffic Load, with Different Numbers of Agents

In the simulation model presented, the process involves a number of local self-organizing agents attached to each grid site (one agent per site). The agents travel through the wide area network by exploiting P2P interconnections; the role of each agent is to gather information about the local resource capacity of the site is attached to. Such logical reorganization of resources is exploited by the schedulers' resource selection scheme, which collects and ranks a large number of resources having the desired characteristics with respect to the user job resource requirement needs. Simulation analysis shows that, as the reorganization or classification of resources proceeds, it allows agents to discover more and more suitable resources in a shorter amount of time, without increasing the traffic load experienced by Grid hosts.

5.9 Discussion of the Results in this Chapter

In this chapter, the proposed schedule-based solution was experimentally evaluated, using different data sets and different scheduling policies. All used data sets, and experiment-related setup information have been described. Then, four major experiments were presented. First, the basic evaluation of the proposed resource characterization method have been performed, while demonstrating the importance of providing resource configuration information to the scheduler, for more realistic scheduling processing. Next, the suitability of our scheduling solution was shown when simulating problems involving job of different application flow types, including a) compute intensive application and b) data intensive applications. Also, the suitability of applied incremental approach was shown here. The third experiment involving various processing time estimates and resource utilization capability of the proposed system was demonstrated. Finally, the last experiment demonstrated the performance evaluation between agent based scheduling and non agent based scheduling. To conclude, the proposed scheduling approach performed very well with respect to various state-of-the-art queue-based solutions. The success of the solution is based on the ability of intelligent agents to coordinate the entire scheduling process, through an optimization routine that uses evaluation to guide the agents, toward good solutions with respect to applied scheduling policies.

CHAPTER SIX

SUMMARY, CONCLUSION AND RECOMMENDATIONS FOR FUTURE DIRECTION

6.1 Summary

The main goal of this dissertation has been to improve the state-of-the-art in multi-component applications scheduling, in order to obtain good speedups for both compute-intensive and data-intensive parallel applications on distributed heterogeneous computing environments. The dissertation started with a detailed description of the complex Grid scheduling problem followed by an overview of related work, that covered several widely used classical scheduling approaches as well as the description of few intelligent multi-agent based scheduling systems. Next, a new scheduling model was proposed to solve the complex Grid scheduling problem. A conceptual architecture design framework for multi-component based scheduling system was presented. The proposed scheduling framework is composed of different scheduling modules that aim at providing solutions to solve complex Grid scheduling problems. The extension of the proposed framework is made easier since it is modelled from the perspective of core object-oriented design paradigm. It was shown how the framework operates and used as reference architecture, starting from a user submitting an application, to scheduler selecting and allocating the best candidate resource.

6.2 Conclusion

The success of the proposed solution is based on several principles and approaches. First of all, the use of intelligent agents is very suitable when dealing with multiple objectives and dynamic environments, since agents adapt very quickly to most dynamic, unpredictable, and highly unreliable distributed environments. In multi-agent

environments, computational resources are distributed across a network of interconnected agents. When compared to a centralized system, multi-agents do not suffer from the single point of failure problem prevalent in most centralized systems. Multi-agents have less performance bottlenecks. In addition, multi-agents efficiently retrieve, filter, and globally coordinate information from sources that are spatially distributed as applicable to the proposed scheduling environment.

In the course of implementing the proposed model, two other primary focuses were given adequate attention:

a. The development of a resource selection strategy, that is required for the scheduling of multi-component applications. These types of applications would usually seek for either schedulable, fixed or both available multi-component heterogeneous resources.
b. Proposing two scheduling policies that are adaptive to the characteristics of heterogeneous distributed system (in this case multi-component systems), considering the problem from the perspective of the multi-component resources and the application user who consumes it.

The intent of these aspects of the work is to enable the proposed model to deal with several real-life situations such as dynamic job arrival, specific job requirements, maintaining communication links between interactive tasks, optimal resource utilization, and minimizing execution time of user applications.

The proposed neural network based multi-agent scheduling approach has been evaluated experimentally, demonstrating that all important characteristics of the studied problem are properly maintained. Also, the performance of the proposed scheduling models has been analysed with respect to several state-of-the-art scheduling algorithms. It has been

shown that the proposed solution usually outperforms all considered heuristic-based scheduling techniques. When multiple execution sites were used for execution of interactive, compute-intensive and data-intensive tasks, our solution performed best in most situations. On the other hand, when the applications are confined to run on single site, the proposed solution performed similarly to the best heuristic-based algorithmic solution, because the lack of sufficient application and system resource cost information seriously limited the capabilities of optimization procedures.

The proposed resource selection policies, based on our evaluation, were shown to be more flexible and provided the required fine-grained dynamic resource configuration information much needed by the multi-component schedulers to make more appropriate scheduling decisions. The advantage of the proposed selection approach is that, rescheduling of user application in the course of machine failure would not be necessary, if on the first instance, careful selection of available and qualified resources that satisfy user application resource requirement were made. The overheads of rescheduling can be very high, for instance, monitoring for and evaluating the need to reschedule to fulfil a given performance contract is a very complex process. When a rescheduling event is initiated, migration of application processes or reallocation of data can be very expensive operations, since the rescheduler must also consider a) the cost of moving the application to a new execution schedule and b) the amount of work remaining in the application that can benefit from a new schedule. Therefore, without careful design, rescheduling can hurt application performance as shown in Fig. 5.20 and discussed in section 5.6.3.

6.3 Recommendations for Future Direction

In the future it is planned to expand the current work in the following directions:

a. Future work will focus on the fully functional prototype implementation of the proposed framework, to see how it adequately adapts to different applications and computational environments. And subsequently, validating the proposed scheduling model in a real-time distributed heterogeneous resource environment.

b. The practical implementation of the MAS scheduling policies in real-life scheduling scenarios would be of interest, to see how it adequately adapts to different multi-component applications. The proposed policies also need to be tested for robustness in terms of scheduling efficiency with regards to application and resource optimal throughput, and utilization under different conditions of uncertainty and heterogeneity of the distributed system resource environment.

c. In our resource management system, resource selection strategy is usually generic, aimed at handling a wide array of applications and does not take into consideration specific application resource requirements. As such an intelligent method for selecting the best resources based on expert knowledge is needed. As part of the dissertation contribution, a Neural Network based Multi-agent resource selection technique capable of mimicking the services of an expert user was proposed. The Neural Network based scheduling framework combined with Multi-agent intelligence is seen as a unique approach, to efficiently deal with the resource selection problem. However, an alternative non-linear resource scheduling technique can be exploited, using fuzzy logic based implementation. Fuzzy logic rules can also be used to characterize the relationship between parallel application resource requirements, and resource scheduling. Another

area of focus can be to introduce reinforcement learning mechanism for the agent based resource scheduling system. The training pattern for agents currently employed is based on a supervised learning algorithm which requires the labelling of training data. However, this may not be always feasible as labelled data may be unavailable at times. Therefore, the incorporation of reinforcement learning with agent networks will allow the learning to be performed, through continuous interaction with the target distributed system environment.

d. Further, applications' characteristics outside the ones covered in our implementation can be considered carefully to figure out more correct and efficient scheduling. For example, given traditional hard disk spinning plate, I/O access pattern plays a significant role that impacts the available I/O bandwidth, as random access pattern results in much lower I/O bandwidth than that of sequential.

List of Journal Publications

Journal Publications

1. Ezugwu, A. E., Buhari, S. M., and Junaidu, S. B. (2014). Resource management system for scientific virtual laboratory applications. *International Journal of Grid and Utility Computing*, 6(1), 8-20.

2. Absalom, E. E., Seyed, B. M., Afolayan, O. A., and Sahalu, J. B. (2013). A Generic Reference Architecture for Collaboratory Scientific Virtual Laboratory. *International Journal of Grid and High Performance Computing (IJGHPC)*, 5(1), 37-52.

3. Ezugwu E. Absalom, Buhari M. Seyed, Junaidu B. Sahalu, (2013).Virtual Machine Allocation in Cloud Computing Environment", *International Journal of Cloud Applications and Computing*, 3(2), pp.47-60.

4. Ezugwu, A. E., Frincu, M. E., & Junaidu, S. B. (2015). A Multiagent-Based Approach to Scheduling of Multi-component Applications in Distributed Systems. In *Artificial Intelligence Perspectives and Applications*, Vol. 347, 2, 2015 (pp.1-12). Springer International Publishing.

5. Ezugwu, A. E., Frincu, M. E., Obiniyi, A. A., Buhari, S. M., and Junaidu, S. B. (2015). Multiagent-based approach for scheduling meta-applications in heterogeneous grid environments, *Multiagent and Grid Systems – An International Journal* (MAGS). 11(2), pp.1-21. IOS Press.

Journal Papers in Press (Springer)

1. Absalom E. Ezugwu, Marc E. Frincu, Sahalu B. Junaidu. (2015). Characterization of Grid Resources Using Measurement-Based Evaluation, *International Transaction on System Science and Applications*, 2015 (in press).

2. Absalom E. Ezugwu, Marc E. Frincu, Seyed M. Buhari, Sahalu B. Junaidu. (2015) Neural Network Based Multi-Agent Approach for Scheduling in Distributed Systems, *International Transaction on System Science and Applications,* 2015 (in press).

3. Absalom E. Ezugwu, Marc E. Frincu , Seyed M. Buhari, Sahalu B. Junaidu. (2015). Multi-Component Applications Scheduling: An Architectural Pattern, *International Journal of Grid and High Performance Computing (IJGHPC),* 7(3), xxx-xxx. (In press).

Conference papers (IEEE)

1. Ezugwu, A. E., Frincu, M. E., and Junaidu, S. B. (2014). Performance characterization of heterogeneous distributed commodity cluster resources. In *Adaptive Science & Technology (ICAST),* 2014 IEEE 6th International Conference on (pp. 1-8). IEEE.

2. Ezugwu, A. E., Frincu, M. E., and Junaidu, S. B. (2014, December). A Reference Architectural Pattern: Component-Based Scheduling System for Heterogeneous Computing Environment. *In Information and Computer Technology (GOCICT),* 2014 Annual Global Online Conference on (pp. 76-84). IEEE.

REFERENCES

Abdullah, M. S., Kimble, C., Benest, I., and Paige, R. (2006). Knowledge-based systems: a re-evaluation. *Journal of Knowledge Management*: 10(3), pp.127-142.

Ali, H. A., and Farrag, T. A. (2006). A Dynamic Ranking Value Guided Scheduling Mechanism (RVGSM) for Mobile Grid. In *Computer Engineering and Systems, the 2006 International Conference on IEEE*. (pp. 368-373).

Alonso, R. S., Tapia, D. I., Bajo, J., García, Ó., de Paz, J. F., and Corchado, J. M. (2013). Implementing a Hardware-Embedded Reactive Agents Platform based on a Service-Oriented Architecture over Heterogeneous Wireless Sensor Networks. *Ad Hoc Networks*, 11 (2013) pp.151–166.

Anagnostopoulos, G. G., and Burlando, P. (2012). An Object-oriented computational framework for the simulation of variably saturated flow in soils, using a reduced complexity model. *Environmental Modelling & Software*, 38, pp.191-202.

Anderson, D. (2004). BOINC: A System for Public-Resource Computing and Storage, in: *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing, IEEE Computer Society*, 2004, pp.4–10.

Anderson, D. and McNeil, G. (1992). Artificial Neural Networks Technology, *Data and Analysis Center for Software*. Rome, NY 13441-4909.

Arnaout, J. P., Musa, R., and Rabadi, G. (2014). A two-stage Ant Colony optimization algorithm to minimize the makespan on unrelated parallel machines—part II: enhancements and experimentations. *Journal of Intelligent Manufacturing*, 25(1), pp.43-53.

Attiya, H., and Welch, J. (2004). Distributed Computing: Fundamentals, Simulations, and Advanced Topics. Wiley, second edition, 2004. On-line version: Retrieved March 28, 2014, from http://dx.doi.org/10.1002/0471478210.

Augonnet, C., Thibault, S., Namyst, R., & Wacrenier, P. A. (2011). StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience*, 23(2), pp.187-198.

Bahnasawy, N. A., Omara, F., Koutb, M. A., and Mosa, M. (2011). Optimization procedure for algorithms of task scheduling in high performance heterogeneous distributed computing systems. *Egyptian Informatics Journal*, 12, 219-229.

Bai, X., Yu, H., Wang, G., Ji, Y., Marinescu, M. G., Marinescu, C. D., and Bölöni, L. (2005). Coordination in Intelligent Grid Environments, in *Proceedings of the IEEE*, on Smart Grids, Volume 93, Issue 3, Pages 613-630.

Baker, A.D. (1991). *Manufacturing Control with a Market-Driven Contract Net*, PhD thesis, Electrical, Computer, and Systems Eng. Dept., Rensselaer Polytechnic Inst., Troy, N.Y. 1991.

Balakrishnan, H., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I. (2003). Looking up data in P2P systems. *Communications of the ACM*, 46(2), pp.43-48.

Basu, M. (2009a). Hybridization of artificial immune systems and sequential quadratic programming for dynamic economic dispatch. *Electric Power Components and Systems*, 37(9), pp.1036-1045.

Basu, S., Costa, L., Brasileiro, F., Banerjee, S., Sharma, P., Lee, S.J. (2009b). NodeWiz: Fault-tolerant grid information service, *Peer-to-Peer Networking and Applications 2* pp.348–366. 10.1007/s12083-009-0030-1.

Bauer, B., Müller, J. P., and Odell, J. (2001). Agent UML: A formalism for specifying multiagent interaction. In *Agent-oriented software engineering*, Vol. 1957, pp. 91-103. Springer, Berlin.

Bergenti, F., and Poggi, A. (2000). Exploiting UML in the design of multi-agent systems. In *Engineering Societies in the Agents World* (pp. 106-113). Springer Berlin/Heidelberg.

Bhama, P. R. K. S., and Selvi, T. S., (2011), "Schedule Based Processor Co allocation in Multiclusters for Scheduling in Grid", *European Journal of Scientific Research*, Vol.55 No.4 (2011), pp.487-499.

Blair, G. S., Coulson, G., Robin, P., & Papathomas, M. (2009, November). An architecture for next generation middleware. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, pp. 191-206. Springer-Verlag.

Blanc, M., Lalande, F. J., (2013), "Improving Mandatory Access Control for HPC clusters", *Future Generation Computer Systems* 29 (2013) pp.876–885.

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. *Computer networks and ISDN systems*, 30(1), 107-117.

Brucker, P. (2007). *Scheduling algorithms*. 5th edition, Springer.

Bussmann, S. (1998). An agent-oriented architecture for holonic manufacturing control. presented at *First Open Workshop, IMS Europe, Lausanne, CH*, 1998.

Buyya, R., Abramson, D., and Giddy, J. (2000) Nimrod/g: an architecture for a resource management and scheduling system in a global computational grid, *Proceedings of the 4th International Conference on High Performance Computing in Asia-Pacific Region, IEEE Computer Press*, pp.283-289.

Buyya, R., Yeo, S. C., Venugopal, S., Broberg, J., Brandic, I. (2009). Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility, *Future Generation Computer Systems* 25 (2009), pp.599–616.

Caminero, A., Rana, O., Caminero, B., Carrión, C., (2011) "Network-aware heuristics for inter-domain meta-scheduling in Grids", *Journal of Computer and System Sciences* 77 (2011), pp.262–281.

Cao, J., Spooner, P. D., Jarvis, A. S., and Nudd, R.G., (2005). "Grid load balancing using intelligent agents", *Future Generation Computer Systems* 21, No. 1, pp.135-149.

Carsten, E., Volker H., and Ramin Y. (2004). Benefits of global Grid computing for job scheduling. In GRID '04: *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 374-379. IEEE, 2004.

Casanova, H., and Dongarra, J., (1997) Netsolve: A network server for solving computational science problems, *Intl. Journal of Supercomputing Applications and High Performance Computing* 11, no. 3, pp.212-223.

Celaya, J., & Arronategui, U. (2012). A task routing approach to large-scale scheduling. *Future Generation Computer Systems*, doi:10.1016/j.future.2012.12.009.

Chapin, S. J., Katramatos, D., Karpovich, J., and Grimshaw, A. (1999). Resource management in legion, *Future Generation Computer Systems* 15, no. 5, pp.583-594.

Chen, R. M., Lo, S. T., and Huang, Y. M. (2007). Combining competitive scheme with slack neurons to solve real-time job scheduling problem. *Expert Systems with Applications*, 33(1), pp.75-85.

Chen, S. R., Tu, R. M., (2009), Development of an agent-based system for manufacturing control and coordination with ontology and RFID technology, *Expert Systems with Applications* 36 (4) (2009) pp.7581–7593.

Chen, V., Barton, R, Allen, J. (2003). COSMOS Technical Report: A review of design and modeling in computer experiments, *Article published in Handbook of Statistics* 22, pp.231–261. Elsevier Science.

Chiang, M., Low, H. S., Calderbank, R. A., and Doyle, C. J. (2007), "Layering as optimization decomposition: A mathematical theory of network architectures." *Proceedings of the IEEE*, January 2007. To appear.

Christodoulopoulos, K., Sourlas, V., Mpakolas, I., and Varvarigos, E. (2009). A comparison of centralized and distributed meta-scheduling architectures for computation and communication tasks in Grid networks. *Computer Communications*, 32(7), pp.1172-1184.

Corman, F., D'ariano, A., Pacciarelli, D., & Pranzo, M. (2014). Dispatching and coordination in multi-area railway traffic management. *Computers and Operations Research*, 44, pp. 146-160.

Corrêa da Silva, F. S., Fernández Venero, M. L., David, D. M., Saleem, M., and Chung, P. W. (2013). Interaction Protocols for Cross-organisational Workflows. *Knowledge-Based Systems* 37 (2013) pp.121–136.

Coulouris, G. F., Dollimore, J., and Kindberg, T. (2005). *Distributed systems: concepts and design*. pearson education.

Coulouris, G., Dollimore, J., Kinfberg, T., and Blair, G., (2012); "*Distributed Systems: Design and Concepts*", fifth edition, Pearson Education, Inc., publishing as Addison-Wesley.

Crespo, A., and Garcia-Molina, H. (2002). Routing indices for peer-to-peer systems. In Distributed Computing Systems, 2002. Proceedings. *22nd International Conference on (pp. 23-32). IEEE*.

Crespo, A., and Garcia-Molina, H., (2005). "Semantic overlay networks for p2p systems." In *Proceedings of the 3rd International Workshop on Agents and Peer-to-Peer Computing*, 1, Springer, New York, NY, USA. pp.33-65.

Czajkowski, K., Foster, I., and Kesseleman, C., (1999), "Resource Co-Allocation in computational Grids", *Proceedings of Eighth IEEE International symposium on High Performance Distributed Computing (HPDC-8)*, pp.219-228.

Da Costa, G., Dias De Assuncao, M., Gelas, J.-P., Georgiou, Y., Lefèvre, L., Orgerie, A.-C., Pierson, J.M., Richard, O., Sayah, A. (2010). Multi-facet approach to reduce energy consumption in clouds and grids: the GREEN-NET framework, in: *ACM/IEEE*

International Conference on Energy-Efficient Computing and Networking, e-Energy 2010, ACM, Passau, Germany, pp.95–104.

Dail, H., Obertelli, G., and Berman, F. (200). Application-Aware Scheduling of a Magnetohydrodynamics Applications in the Legion Metasystem. In *Heterogeneous Computing Workshop, 2000.(HCW 2000) Proceedings. 9th (pp. 216-228). IEEE*.

Dalfard, V. M., and Mohammadi, G. (2012). Two meta-heuristic algorithms for solving multi-objective flexible job-shop scheduling with parallel machine and maintenance constraints. *Computers & Mathematics with Applications*, 64(6), pp.2111-2117.

Darmoul, S., Pierreval, H., and Hajri–Gabouj, S. (2013). Handling disruptions in manufacturing systems: An immune perspective. *Engineering Applications of Artificial Intelligence,* 26 (2013) pp.110-121.

Darriba, D., Taboada, G. L., Doallo, R., and Posada, D. (2012). jModelTest 2: more models, new heuristics and parallel computing. *Nature Methods*, 9(8), pp.772-772.

David, O., Ascough, J. C., Lloyd, W., Green, T. R., Rojas, K. W., Leavesley, G. H., and Ahuja, L. R. (2013). A software engineering perspective on environmental modeling framework design: *The Object Modeling System. Environmental Modelling & Software,* 39 (2013) pp.201-213.

Davis, R. I., and Burns, A. (2005). Hierarchical fixed priority pre-emptive scheduling. In *Real-Time Systems Symposium, 2005. RTSS 2005. 26th IEEE International*, pp. 10-pp. IEEE.

Dean, J., and Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1), pp.107-113.

Decker, K. S., and Lesser, V. R. (1992). Generalizing the partial global planning algorithm. *International Journal of Intelligent and Cooperative Information Systems*, 1(02), 319-346.

Donassolo, B., Legrand, A., and Geyer, C. (2011). Non-cooperative scheduling considered harmful in collaborative volunteer computing environments. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on IEEE*. pp. 144-153.

Dong, F., and Akl, S. G. (2006). *Scheduling algorithms for grid computing: State of the art and open problems* (Vol. 504). Technical report.

Douglas, C. (2008). *Design and analysis of experiments*. John Wiley & Sons.

Dror, G. F. (2005). Experimental analysis of the root causes of performance evaluation results: A backfilling case study. *IEEE Transactions on Parallel and Distributed Systems*, 16(2):175-182.

Dror, G. F., Larry R., Uwe S., Kenneth C. S., and Parkson W. (1997). Theory and practice in parallel job scheduling. In Dror G. Feitelson and Larry Rudolph, editors, Job Scheduling Strategies for Parallel Processing, volume 1291 of *LNCS*, pages 1-34. Springer Verlag.

Drost, N., van Nieuwpoort, R. V., Maassen, J., Seinstra, F. J., and Bal, H. E. (2011). Zorilla: a peer-to-peer middleware for real-world distributed systems. *Concurrency and Computation: Practice and Experience*, 23(13), pp.1506-1521.

Durfee, E. H., (1999) "Distributed Problem Solving and Planning." *Lecture Notes in Computer Science*, MIT Press, pp.1-34.

Durfee, E. H., and Lesser, V. R. (1991). Partial global planning: A coordination framework for distributed hypothesis formation. *Systems, Man and Cybernetics, IEEE Transactions* on, 21(5), 1167-1183.

Durfee, E. H.: Distributed Problem Solving and Planning. (2014). *Lecture Notes in Computer Science*, MIT Press, 2014; pp1-34.

Ebrahimi, B. P., Bertels, K., Vassiliadis, S., and Sigdel, K. (2004). Matchmaking within multi-agent systems. *Proceeding of ProRisc-2004*.

Edelkamp, S., Leue, S., and Lluch-Lafuente, A. (2004). Directed explicit-state model checking in the validation of communication protocols. *International journal on software tools for technology transfer*, 5(2-3), 247-267.

Eliassen, F. (2010). "*Introduction to Distributed System*," Lecture note:

Fatima, S. S., & Wooldridge, M. (2001, May). Adaptive task resources allocation in multi-agent systems. *In Proceedings of the fifth international conference on Autonomous agents* (pp. 537-544). ACM.

Foster, I., Kesselman C., Tsudik G., and Tuecke S.,(1998). A security architecture for computational Grids, *ACM Conference on Computers and Security*, pp. 83–91.

Foster, I. (2006). Globus toolkit version 4: Software for service-oriented systems. *Journal of computer science and technology*, 21(4), pp.513-520.

Foster, I., Zhao, Y., Raicu, I., & Lu, S. (2008). Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08, IEEE*, pp. 1-10.

Foster, I. (2006). Globus Toolkit Version 4: Software for Service-Oriented Systems, *Journal of Computer Science and Technology* 21, pp.513–520.

Frey, J.,Tannenbaum T., Livny M., Foster F., Tuecke S. (2002), "Condor-G: A Computation Management Agent for Multi-Institutional Grids", *Cluster Computing* 2002, 5(3), pp. 237–246.

Frincu, E. M., (2011), *Adaptive Scheduling for Distributed Systems*, Ph.D dissertation, West University of Timisoara (unpublished).

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1993). *Design patterns: Abstraction and reuse of object-oriented design*. Springer Berlin Heidelberg, pp.406-431.

Gawiejnowicz, S., and Kononov, A. (2014). Isomorphic scheduling problems. *Annals of Operations Research*, 213(1), pp.131-145.

Gimenes, I. M., Tanaka, S. A., and de Oliveira, J. P. M. (2000). An object oriented framework for task scheduling. In *Technology of Object-Oriented Languages, 2000. TOOLS 33. Proceedings. 33rd International Conference on IEEE*. pp. 383-394.

González, M., Moreno, L., and Martínez, P. (2012). An Approach to user Interface Design of an Accessible user Agent. *Procedia Computer Science*, 14, pp.254-262.

Goodrich, M. T., and Tamassia R. (2006), *Data Structures and Algorithms in Java*, 4th Edition, John Wiley & Sons, Inc.

Gopalan, K., & Chiueh, T. C. (2001). Multiresource allocation and scheduling for periodic soft real-time applications. *In Electronic Imaging 2002, International Society for Optics and Photonics*. pp. 34-45.

Graham, R. J., (2001). *Real-time scheduling in distributed multi-agent system*, PhD thesis, University of Delaware. pp.2.

Gupta, A., & Milojicic, D. (2011, October). Evaluation of hpc applications on cloud. In *Open Cirrus Summit (OCS), 2011 Sixth, IEEE*. pp. 22-26.

Hadim, S., & Mohamed, N. (2006). Middleware: Middleware challenges and approaches for wireless sensor networks. *IEEE distributed systems online*, 7(3), 1.

Hai, J., Buyya, R., and Baker, M. (2001). Cluster Computing Tools, Applications, and Australian Initiatives for Low Cost Supercomputing. *Journal of High Performance Computing (2001)*, pp1-11.

Hamid, N., Haron, F., and Yong, C. H. (2006). Resource Discovery Using PageRank Technique in Grid Environment. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium* on (Vol. 1, pp. 135-140). IEEE.

Hao, W., Yang, Y., & Lin, C. (2007). Qos Performance Analysis for Grid Services Dynamic Scheduling System. In *Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on IEEE*. (pp. 2012-2015).

He, Q., Zhou, S., Kobler, B., Duffy, D., and McGlynn, T. (2010, June). Case study for running HPC applications in public clouds. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing* (pp. 395-401). ACM.

Hindman, B., Konwinski, A., Zaharia, M., and Stoica, I., (2009). A Common Substrate for Cluster Computing. In *Workshop on Hot Topics in Cloud Computing (HotCloud)* Vol.2009.

Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R., and Stoica, I. (2011). Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation (pp. 22-22). USENIX Association*.

Hooda, M., & Bhadauria, M. (2013). Recent innovations in Distributed Systems: Challenges and Benefits. *International Journal of Computers & Technology*, 10(10), pp.2057-2061.

Hsu, C. H., & Feng, W. C. (2005). A power-aware run-time system for high-performance computing. *In Proceedings of the 2005 ACM/IEEE conference on Supercomputing* (p. 1). IEEE Computer Society.

Iamnitchi, A., & Foster, I. (2000). A problem-specific fault-tolerance mechanism for asynchronous, distributed systems. In *Parallel Processing, 2000. Proceedings. 2000 International Conference on IEEE*. (pp. 4-13).

Iglesias, C., Garijo, M., and Gonzalez, J. (1999). A survey of agent-oriented methodologies. Intelligent Agents V: *Agents Theories, Architectures, and Languages*, pp.317-330.

Imamagic, E., Radic B., Dobrenic D., (2006), "An Approach to Grid Scheduling by using CondorG Matchmaking Mechanism", *Journal of Computing and Information Technology– CIT*, 329–336, Doi:10.2498/cit.2006.04.09.

Isaac, C.. (2014). *Agent Grid Sim.* Retrieved January 28, 2014, from https://sourceforge.net/projects/agentGridrepast

Iyer, A., & Marculescu, D. (2002). Power and performance evaluation of globally asynchronous locally synchronous processors. In *Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on IEEE*. pp. 158-168.

Jackson, J. P. (2012). *Constrained Task Assignment and Scheduling on Networks of Arbitrary Topology*, Ph.D Doctorial dissertation, University of Michigan.

Janjic, V. (2012). *Load balancing of irregular parallel applications on heterogeneous computing environments* (Doctoral dissertation, University of St Andrews).

Jennings, N. R. (2001). An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4), pp.35-41.

Johnson, R. E., (1997). Components, frameworks, patterns. *ACM SIGSOFT Software Engineering Notes*, 22(3), pp.10-17.

Karatza, H. (2004a). Scheduling in distributed systems. In *Performance Tools and Applications to Networked Systems* (pp. 336-356). Springer Berlin Heidelberg..

Karatza, H. D. (2004b). Simulation study of multitasking in distributed server systems with variable workload. *Simulation Modelling Practice and Theory*, 12(7), 591-608.

Kavi, K., Kung, D., Bhambhani, H., Pancholi, G., Kanikarla, M., and Shah, R. (2003). Extending UML to modeling and design of multi-agent systems. In *Proc. of ICSE 2003 Workshop on Software Engieering for Large Multi-Agent Systems (SELMAS), Portland, Oregon*.

Kendall, E. A., Malkoun, M. T., and Jiang, C. (1997). Multi-agent system design based on object-oriented patterns. *JOOP*, 10(3), pp.41-47.

Khoo, B. T., Veeravalli, B., Hung, T., & Simon See, C. W. (2007). A multi-dimensional scheduling scheme in a Grid computing environment. *Journal of Parallel and Distributed Computing*, 67(6), pp.659-673.

Kim, J. S., Nam, B., Keleher, P., Marsh, M., Bhattacharjee, B., Sussman, A. (2008). Trade-offs in Matching Jobs and Balancing Load for Distributed Desktop Grids. *Future Generation Computer Systems* 24, pp.415–424.

Klusácek, D. and Rudová H. (2010). Alea 2 - Job Scheduling Simulator. In *proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools 2010), ICST*.

Klusacek, D., Matyska, L., Rudova, H., Baraglia, R., and Capannini, G., (2007), "Local Search for Grid Scheduling." In *Doctoral Consortium at the International Conference on Automated Planning and Scheduling (ICAPS'07)*, USA.

Konwinski, D. A. (2012), *Multi-agent Cluster Scheduling for Scalability and Flexibility*, Ph.D dissertation, University of California, Berkeley, USA.

Krishnan, R., Smith, M. D., and Telang, R. (2003), "The economics of peer-to-peer networks", *Journal of Information Technology Theory and Application*, 5, 3, 2003, pp.1-6.

Krueger, C.W., (1992). *Software reuse. ACM Comput*. Surveys 24, 2, pp.131–183.

Kumar, C., Altinkemer, K., and De, P., (2011), "A mechanism for pricing and resource allocation in peer-to-peer networks", *Electronic Commerce Research and Applications* 10 (2011), pp.26–37.

Kumar, S., Sharma, V. K., and Kumari, R. (2014). Randomized Memetic Artificial Bee Colony Algorithm. *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*. In Press.

Kumar, V., Grama, A., Gupta, A., and Karypis, G. (1994). *Introduction to parallel computing* (Vol. 110). Redwood City: Benjamin/Cummings Publishing Company.

Lee, E. A. (2008). Cyber physical systems: Design challenges. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on IEEE*. pp. 363-369.

Lejeune, A., Béchet, F., Boudaoud, H., Mathieu, N., and Potier-Ferry, M. (2012). Object-oriented design to automate a high order non-linear solver based on asymptotic numerical method. *Advances in Engineering Software*, 48, pp.70-88.

Li, C., and Li, L. (2012). Design and implementation of economics-based resource management system in ad hoc grid. *Advances in Engineering Software* 45 (2012) pp.281–291.

Li, Y., Tan, Y., and Zhou, Y., (2002), On the scale of peer-to-peer networks. In *Proceedings of 12th Annual Workshop on Information Technology and Systems, Barcelona, Spain*, 2002.

Lin, G.Y.-J., and Solberg, J. J.(1992) Integrated Shop Floor Control Using Autonomous Agents, *IIE Trans.: Design and Manufacturing*, vol.24, no. 3, July 1992, pp. 57–71.

Logenthiran, T. (2012). *Multi-agent system for control and management of distributed power systems*, PhD thesis, National University of Singapore, pp.18.

Logenthiran, T., Srinivasan, D., and Khambadkone, M. A., (2011). Multi-agent system for energy resource scheduling of integrated microgrids in a distributed system. *Electrical Power System Research*, vol.81, no.1, pp.138-148.

Logenthiran, T., Srinivasan, D., Khambadkone, M. A., and Sundar Raj, T. (2010). Optimal sizing of an islanded microgrid using Evolutionary Strategy, *IEEE International Conference on Probabilistic Methods Applied to Power Systems*, pp.12-17, Singapore, pp.14-17.

Long, Q., Lin, J., and Sun, Z. (2011). Agent scheduling model for adaptive dynamic load balancing in agent-based distributed simulations. *Simulation Modelling Practice and Theory*, 19(4), pp.1021-1034.

Lucas-Simarro, J. L., Moreno-Vozmediano, R., Montero, R. S., and Llorente, I. M. (2013). Scheduling strategies for optimal service deployment across multiple clouds. *Future Generation Computer Systems*, 29(6), pp.1431-1441.

Lucena, C., and Nunes, I. (2013). Contributions to the emergence and consolidation of Agent-oriented Software Engineering. *Journal of Systems and Software*, 86(4), pp.890-904.

Luther, A., Buyya, R., Ranjan, R., and Venugopal, S. (2005). Peer-to-Peer Grid Computing and a .NET-based Alchemi Framework, In *High Performance Computing: Paradigm and Infrastructure* , L Yang and M. Guo (eds), Wiley Press, New Jersey, USA.

Lynch, S. and Rajendran, K. (2004). Design Diagrams for Multi-Agent Systems, *16th Workshop of the Psychology of Programming Interest Group. Carlow, Ireland* pp66-78.

Lynden, S., and Rana, F. O.(2002). Coordinated Learning to support Resource Management in Computational Grids. *2nd IEEE International Conference on Peer-2-Peer Computing, September 2002, Linkoping, Sweden. IEEE Computer Society Press*.

Maier, D. (2001). Object-Oriented Database Theory. An Introduction and Indexing in OODBS. White Paper.

Malathi, M. G., and Sarumathi, M. S. (2010). Survey on GRID Scheduling. *Journal of Computer Applications*, 3(3), 22.

Marinescu, F., (2002). *EJB design patterns* New York: Wiley, pp. 70-75.

Martínez, P. L., Barros, L., and Drake, J. M. (2012). Design of component-based real-time applications. *The Journal of Systems and Software* 86 (2013) pp.449– 467.

Masri, S. F., Bekey, G. A., and Safford, F. B. (1980). A global optimization algorithm using adaptive random search. *Applied mathematics and computation*, 1980; 7(4), 353-375.

Massie, M. L., Chun, B. N., & Culler, D. E. (2004). The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7), pp.817-840.

Mee, R. (2009). A Comprehensive Guide to Factorial Two-Level Experimentation. *Springer, New York.*

Mesnier, M., Ganger, G. R., Riedel, E., (2003). "Object-Base Storage", *IEEE Communications Magazine*. pp85-90.

MetaCentrum. (2014). Czech National Grid Infrastructure, Retrieved April 23, 2014 from http://www.metacentrum.cz/en/.

Metaxiotis, K. S., Askounis, D., and Psarras, J. (2002). Expert systems in production planning and scheduling: A state-of-the-art survey. *Journal of Intelligent Manufacturing*, 13(4), pp.253-260.

Mihail, M. (2012). "*Decentralized Coordination in Multi-Agent Systems*" PhD Thesis, Vrije Universiteit Brussel.

Mihaila, C., Mihaili, A. (2008). Evolutionary Algorithm for Uniform Parallel Machines Scheduling, in *IEEE Processdings of the European Modelling Symposium*, pp.76-80, Liverpool, United Kingdom.

Mihaila, C. (2011). *Evolutionary Computing in Scheduling*. PhD Thesis, Babes-Bolyai University.

Milojicic, S. D., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S., and Xu, Z., (2003), "Peer-to-peer computing", *Tech. report, HP Laboraties Palo Alto*, http://www.hpl.hp.com/techreports/2002/HPL-2002-57R1.pdf.

Mishra, R. K., & Lohani, B. (2007). An Object-Oriented Software Development Approach to Design Simulator for Airborne Altimetric Lidar. In *National conference on Emerging Trends in Information Technology*.

Mohamed, H. H., & Epema, D. H. (2004). An evaluation of the close-to-files processor and data co-allocation policy in multiclusters. In *Cluster Computing, 2004 IEEE International Conference on (pp. 287-298). IEEE*.

Monroe, R. T., Kompanek, A., Melton, R., and Garlan, D. B. (1996). Architectural styles, design patterns, and objects. *IEEE software*, 43.

Moschakis, A. I., and Karatza, D. H., (2010), "Evaluation of gang scheduling performance and cost in a cloud computing system" *J Supercomput*, DOI 10.1007/s11227-010-0481-4, Springer, Berlin, September, 2010, pp.1-18.

Nadiminti, K., De Assunção, M. D., and Buyya, R. (2006). Distributed systems and recent innovations: Challenges and benefits. *InfoNet Magazine*, 16(3), pp.1-5.

Naor, M., and Wieder, U., (2003). A simple fault tolerant distributed hash table. In *Peer-to-Peer Systems II*, Springer Berlin Heidelberg, pp.88-97.

Noronha, S. J., and Sarma, V. V. S. (1991). Knowledge-based approaches for scheduling problems: A survey. *Knowledge and Data Engineering, IEEE Transactions on*, 3(2), pp.160-171.

Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D. (2009). The Eucalyptus Open-Source Cloud-Computing System, in: *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, IEEE Computer Society,2009*, pp.124–131.

Odell, J., Parunak, H. V. D., and Bauer, B. (2000). Extending UML for agents. *Ann Arbor*, pp.1001-48103.

Opdyke, W. F., (1990). Refactoring: An aid in designing application frameworks and evolving object-oriented systems. In *Proc. of 1990 Symposium on Object-Oriented Programming Emphasizing Practical Applications (SOOPPA)*.

Ouelhadj, D., Garibaldi, M. G., MacLaren, J., Sakellariou, R., and Krishnakumar, K., (2005). " A multi-agent infrastructure and a service level agreement negotiation protocol for robust scheduling in grid computing", *EGC*, pp.651-660.

PALAI, (2011). *Object Oriented Modeling and Design*. Lecture Note, Department of Computer Science, SJCET, PALAI. Retrieved April 5, 2013, from

Patzák, B., and Rypl, D. (2012). Object-oriented, parallel finite element framework with dynamic load balancing. *Advances in Engineering Software*, 47(1), pp.35-50.

Pinedo, M., and Yen, B. P. C. (1997). On the design and development of object-oriented scheduling systems. *Annals of Operations Research*, 70, pp.359-378.

Plaza, A., Du, Q., and Chang, Y. L. (2009,). High performance computing for hyperspectral image analysis: perspective and state-of-the-art. In *Geoscience and Remote Sensing Symposium, 2009 IEEE International, IGARSS 2009*, Vol. 5, pp.V-72).

Puppin, D., Moncelli, S., Baraglia, R., Tonellotto, N., Silvestri, F. (2005). A Grid information service based on peer-to-peer, in: *Proc. of the 11th Intl. Euro-Par Conference, Lisbon, Portugal*.

Quinn, M., J. *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill Inc. 2004. ISBN 0-07-058201-7, pp.155-158.

Rădulescu, A., and van Gemund, A. J. (1999). On the complexity of list scheduling algorithms for distributed-memory systems. In *Proceedings of the 13th international conference on Supercomputing (pp. 68-75). ACM*.

Rădulescu, M. (2009), *Object-oriented Database Development Using DB4O*. Masters Thesis, "BABEŞ-BOLYAI" University Cluj-Napoca.

Rahman, M., Ranjan, R., and Buyya, R. (2010). Cooperative and decentralized workflow scheduling in global grids. *Future Generation Computer Systems*, 26(5), pp.753-768.

Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S., (2001). A scalable content-addressable network, *ACM*, Vol. 31, No. 4, pp.161-172.

Ripeanu, M. (2001). Peer-to-peer architecture case study: Gnutella network. Peer-to-Peer Computing, 2001. Proceedings. *First International Conference on IEEE*, pp. 99-100.

Rypl, D., and Patzák, B. (2012). Object oriented implementation of the T-spline based isogeometric analysis. *Advances in Engineering Software* 50, pp.137–149.

Sakellariou, R., and Zhao, H. (2004). A hybrid heuristic for DAG scheduling on heterogeneous systems. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International (p. 111). IEEE*.

Salamon, T. (2011). *Design of Agent-Based Models: Developing Computer Simulations for a Better Understanding of Social Processes*. Academic series. Bruckner Publishing, Repin, Czech Republic.

Sarathy, V., Narayan P., and Mikkilineni R. (2012). *Next generation Cloud Computing Architecture: Enabling real-time dynamism for shared distributed physical infrastructure*. Retrieved December 3, 2013 from

Sauer, J. (2001). Knowledge-based design of scheduling systems. *Intelligent Automation and Soft Computing*, 7(1), pp.55-62.

Sauer, J., and Bruns, R. (1997). Knowledge-based scheduling systems in industry and medicine. *IEEE Expert*, 12(1), pp.24-31.

Schroeder, B., and Gibson, G. A. (2007). Understanding failures in petascale computers. In *Journal of Physics: Conference Series* Vol. 78, No. 1, pp.012022.

Selvi, S. T., Balakrishnan, P., Kumar, R., and Rajendar, K. (2007). Trust based grid scheduling algorithm for commercial grids. In *Conference on Computational Intelligence and Multimedia Applications, 2007. International Conference on* (Vol. 1, pp. 545-551). IEEE.

Shah, S. N. M., Haron, N., Zakaria, M. N. B., and Mahmood, A. K. B. (2012a). Agent-based Robust Grid Scheduling Framework for High Performance Computing. *AASRI Procedia*, 1, pp.554-560.

Shah, S. N. M., Zakaria, M. N. B., Haron, N., Mahmood, A. K. B., and Naono, K. (2012b). Design and Evaluation of Agent Based Prioritized Dynamic Round Robin Scheduling Algorithm on Computational Grids. *AASRI Procedia*, 1, pp.531-543.

Shan, H., Antypas, K., and Shalf, J. (2008, November). Characterizing and predicting the I/O performance of HPC applications using a parameterized synthetic benchmark. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing, IEEE Press*, pp. 42.

Shen, W. (2002). Distributed manufacturing scheduling using intelligent agents. *Intelligent Systems, IEEE*, 17(1), pp.88-94.

Shen, W., and Norrie, D. (2007). Facilitators, Mediators or Autonomous Agents. *Division of Manufacturing, the University of Calgary, Alberta, Canada*.

Shih, W. C., Yang, C. T., and Tseng, S. S. (2007). A performance-based parallel loop scheduling on grid environments. *The Journal of Supercomputing*, Vol. 41, pp.247-267.

Shih, W-C, Yang, C. T, and Tseng, S. S. (2009). Using a Performance-based Skeleton to Implement Divisible Load Applications on Grid Computing Environments. *Journal of Information Science and Engineering* 25, 59-81.

Sinopoli, B., Sharp, C., Schenato, L., Schaffert, S., and Sastry, S. S. (2003). Distributed control applications within sensor networks. *Proceedings of the IEEE*, 91(8), pp.1235-1246.

Sotomayor, B., Montero, S. R., Llorente, M. I., and Foster, I. (2009). Capacity Leasing in Cloud Systems using the OpenNebula Engine, *Cloud Computing and Applications* 2008 (CCA08).

Stavrinides, G. L., & Karatza, H. D. (2012). Scheduling real-time DAGs in heterogeneous clusters by combining imprecise computations and bin packing techniques for the exploitation of schedule holes. *Future Generation Computer Systems*, 28(7), pp.977-988.

Su, W., Sadek, A. K., & Liu, K. R. (2008). Cooperative communication protocols in wireless networks: Performance analysis and optimum power allocation. *Wireless Personal Communications*, 44(2), pp.181-217.

Suárez Barón, S. A. (2011). *Dynamic task allocation and coordination in cooperative multi-agent environments*. Universitat de Girona.

Sulistio, A. Cibej, U., Venugopal, S., Robic, B., and Buyya, R. (2008). A toolkit for modelling and simulating data Grids: an extension to GridSim. *Concurrency and Computation: Practice & Experience*, 20(13):1591-1609.

Sycara, K. P. (1998). Multi-agent systems, *AI Magazine*, 19(2): pp79-92.

Sycara, K., Widoff, S., Klusch, M., and Lu, J. (2002). Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Autonomous agents and multi-agent systems*, 5(2), 173-203.

Tang, L., Liu, W., and Liu, J. (2005). A neural network model and algorithm for the hybrid flow shop scheduling problem in a dynamic environment. *Journal of Intelligent Manufacturing*, 16(3), pp.361-370.

Tannenbaum, T., Wright, D., Miller, K., Livny, M. (2002). Condor: A Distributed 1037 Job Scheduler, *Beowulf cluster computing with Linux*, pp.307–350.

Topcuoglu, H., Hariri, S., and Wu, Y. M. (2002). "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel and Distributed Systems*, vol. 13, no.3, pp. 260-274.

Vaquero, L. M., Rodero-Merino, L., Caceres, J., and Lindner, M. (2008). A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1), pp.50-55.

Venugopal, S., Nadiminti, K., Gibbins, H., and Buyya, R. (2008). Designing a resource broker for heterogeneous grids. *Software: Practice and Experience*, 38(8), pp.793-825.

Verma, A., and Kaushal, S. (2014). Deadline constraint heuristic–based genetic algorithm for workflow scheduling in cloud. *International Journal of Grid and Utility Computing*, 5(2), pp.96-106.

Vlassis, N. (2007). A concise introduction to multiagent systems and distributed artificial intelligence. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 1(1), pp.1-71.

Vokrinek, J., Komenda, A., and Pechoucek, M. (2011). Abstract architecture for task-oriented multi-agent problem solving. Systems, Man, and Cybernetics, Part C: *Applications and Reviews, IEEE Transactions on*, 41(1), pp.31-40.

Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D., and Maltzahn, C. (2006). Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation. USENIX Association*, pp.307-320.

Weiss, G. (Ed.). (1999). Multiagent systems: a modern approach to distributed artificial intelligence. *The MIT press*.

Weissman, B. J. (2000). "Scheduling multi-component applications in heterogeneous wide-area networks." Heterogeneous Computing Workshop, 2000.(HCW 2000) Proceedings. 9th. IEEE, 2000.

Wooldridge, M. (2009). *An Introduction to Multi-agent Systems*. John Wiley and Sons Ltd., Second edition.

Wu, J. C., Poppa, K., Leu, M. C., and Liu, X. F. (2012). Integrated function structure and object-oriented design framework. *Computers in Industry*, Volume 63, Issue 5, June 2012, pp458-470.

Wu, J., Xu, X., Zhang, P., and Liu, C. (2011). A novel multi-agent reinforcement learning approach for job scheduling in Grid computing. *Future Generation Computer Systems*, 27(5), pp.430-439.

Xue, Z., Dong, X., Hu, L., and Li, J. (2012). "A performance and energy optimization mechanism for cooperation-oriented multiple server clusters". *Future Generation Computer Systems*, 28(5), pp.801-810.

Yang, C. T., Shih, W. C., and Tseng, S. S., (2008). Dynamic partitioning of loop iterations on heterogeneous PC clusters. *The Journal of Supercomputing*, Vol. 44, pp. 1-23.

Yang, S. Y. (2013). A Novel Cloud Information Agent System with Web Service Techniques: Example of an Energy-saving Multi-agent system. *Expert Systems with Applications*, 40 (2013) pp.1758–1785.

Yates, F., & Cochran, W. G. (1938). The analysis of groups of experiments. The Journal of Agricultural Science, 28(04), 556-580.

Yim, H., Cho, K., Kim, J., and Park, S. (2000). Architecture-centric object-oriented design method for multi-agent systems. In *Multi-Agent Systems, 2000. Proceedings. Fourth International Conference on IEEE*, pp.469-470.

Yu, J., Buyya, R., and Ramamohanarao, K. (2008). Workflow scheduling algorithms for grid computing. In Metaheuristics *for scheduling in distributed computing environments. Springer Berlin Heidelberg*, pp. 173-214.

Zarandi, H. M., and Ahmadpour, P. (2009), Fuzzy agent-based expert system for steel making process, *Expert Systems with Applications* 36 (5) (2009) pp.9539–9547.

Zhang, X. (2003). *Interactive event-based intelligent scheduling.* Ph.D dissertation, Rand Afrikaans University.

Zhang, D., and Dong, Y. (2000). An efficient algorithm to rank web resources. *Computer Networks*, 33(1), 449-455.

Zhang, F., Cao, J., Li, K., Khan, S. U., & Hwang, K. (2014). Multi-objective scheduling of many tasks in cloud platforms. *Future Generation Computer Systems*, 37(Complete), pp. 309-320.

Zhang, H. B., Tang, L. S., and Liu, L. X. (2009). Survey of grid scheduling. *Computer Engineering and Design*, 9, pp.26.

Zhu, Y., Ni, L. M. (2013). *A Survey on Grid Scheduling Systems*, Technical Report SJTU_CS_TR_200309001, Department of Computer Science and Engineering, Shanghai Jiao Tong University.

APPENDICES

Appendix A: First-Fit Resources Selection and Ranking Method

Job Resource Requirement Table (Jobs with interactive tasks)

Job ID	NP	PS	MS	BW
Job_1	1	2608	2849	70
Job_2	1	2608	2849	70
Job_3	4	8853	8187	200

NP = Number of Processors, PS = Processor Speed, MS = Memory Size, BW = Bandwidth

Below is a description of the resource selection and ranking process implemented using the First Fit resource selection method proposed in this dissertation. There are three stages in the selection process.

1. Resource filtering stage
2. Resource classification according to computing capabilities
3. Selection of the best ranked resource that meets the submitted user job description

Application execution for the proposed system entails the following parameters estimations:

1. Resource requirements of a given application.
2. Computation capability of the available resources.
3. Present state of the resources.
4. User preference based on resource performance function.

All resources values were randomly generated.

	Cluster 1			
C1Node ID	**NP**	**PS**	**MS**	**BW**
node_1	4	8853	8187	200
node_2	10	6832	4052	300
node_3	4	4341	3694	170
node_4	9	4341	6326	140
node_5	5	3661	5335	114
node_6	4	4728	5627	105
node_7	5	5021	7349	169
node_8	10	2488	2352	161
node_9	9	2843	5612	128
node_10	1	3466	3268	156
TOTAL	**61**	**46574**	**51802**	**1643**

	Job to Resource Matching		No Match			
				Matching Scores		
Job 1	**Job2**	**Job3**				
node_1	node_1	node_1	1	1	1
node_2	node_2	No Match	1	1	0
node_3	node_3	No Match	1	1	0
node_4	node_4	No Match	1	1	0
node_5	node_5	No Match	1	1	0
node_6	node_6	No Match	1	1	0
node_7	node_7	No Match	1	1	0
No Match	No Match	No Match	0	0	0
node_9	node_9	No Match	1	1	0
node_10	node_10	No Match	1	1	0

Total Matching Score for Cluster 1 =	**0.90**	**0.90**	**0.10**

First Collection of Qualified Clusters

Qualified Nodes for Job1 in Cluster1				
node_1	4	8853	8187	200
node_2	10	6832	4052	300
node_3	4	4341	3694	170
node_4	9	4341	6326	140
node_5	5	3661	5335	114
node_6	4	4728	5627	105
node_7	5	5021	7349	169
node_9	9	2843	5612	128
node_10	1	3466	3268	156

Cluster 2

C2Node ID	NP	PS	MS	BW
node_1	4	5030	9851	200
node_2	6	6031	7481	250
node_3	10	5806	15400	151
node_4	10	5909	15282	137
node_5	4	5411	6722	176
node_6	4	5662	6537	171
node_7	6	5496	4609	300
node_8	8	5108	13368	189
node_9	8	3838	3423	120
node_10	9	3803	11185	131
TOTAL	**69**	**52094**	**93858**	**1825**

Job 1	Job2	Job3	Matching Scores		
node_1	node_1	No Match	1	1	0
node_2	node_2	No Match	1	1	0
node_3	node_3	No Match	1	1	0
node_4	node_4	No Match	1	1	0
node_5	node_5	No Match	1	1	0
node_6	node_6	No Match	1	1	0
node_7	node_7	No Match	1	1	0
node_8	node_8	No Match	1	1	0
node_9	node_9	No Match	1	1	0
node_10	node_10	No Match	1	1	0
Total Matching Score for Cluster 2 =			**1.00**	**1.00**	**0.00**

Qualified Nodes for Job1 in Cluster2				
node_1	4	5030	9851	200
node_2	6	6031	7481	250
node_3	10	5806	15400	151
node_4	10	5909	15282	137
node_5	4	5411	6722	176
node_6	4	5662	6537	171
node_7	6	5496	4609	300
node_8	8	5108	13368	189
node_9	8	3838	3423	120
node_10	9	3803	11185	131

Cluster 3

C3Node ID	NP	PS	MS	BW
node_1	10	5081	12895	211
node_2	24	7661	10093	170
node_3	20	3027	1545	211
node_4	14	2461	6441	163
node_5	10	5886	7715	223
node_6	6	2751	9280	203
node_7	6	5478	6896	185
node_8	6	5752	10048	195
node_9	14	5871	10969	226
node_10	6	4398	12064	172
TOTAL	**116**	**48366**	**87946**	**1959**

Job 1	Job2	Job3	Matching Scores		
node_1	node_1	No Match	1	1	0
node_2	node_2	No Match	1	1	0
No Match	No Match	No Match	0	0	0
No Match	No Match	No Match	0	0	0
node_5	node_5	No Match	1	1	0
node_6	node_6	No Match	1	1	0
node_7	node_7	No Match	1	1	0
node_8	node_8	No Match	1	1	0
node_9	node_9	No Match	1	1	0
node_10	node_10	No Match	1	1	0
Total Matching Score for Cluster 3 =			**0.80**	**0.80**	**0.00**

Qualified Nodes for Job1 in Cluster3				
node_1	10	5081	12895	211
node_2	24	7661	10093	170
node_5	10	5886	7715	223
node_6	6	2751	9280	203
node_7	6	5478	6896	185
node_8	6	5752	10048	195
node_9	14	5871	10969	226
node_10	6	4398	12064	172

Cluster 4

C4Node ID	NP	PS	MS	BW
node_1	9	5127	1619	236
node_2	4	6067	7270	267
node_3	7	4842	5085	283
node_4	10	3770	2500	207
node_5	8	5596	2955	227
node_6	10	5748	6258	255
node_7	6	2794	2125	256
TOTAL	**54**	**33944**	**27812**	**1731**

Job 1	Job2	Job3	Matching Scores		
No Match	No Match	No Match	0	0	0
node_2	node_2	No Match	1	1	0
node_3	node_3	No Match	1	1	0
No Match	No Match	No Match	0	0	0
node_5	node_5	No Match	1	1	0
node_6	node_6	No Match	1	1	0
No Match	No Match	No Match	0	0	0
Total Matching Score for Cluster 4 =			**0.57**	**0.57**	**0.00**

Qualified Nodes for Job1 in Cluster4				
node_2	4	6067	7270	267
node_3	7	4842	5085	283
node_5	8	5596	2955	227
node_6	10	5748	6258	255

Cluster 5

C5Node ID	NP	PS	MS	BW
node_1	10	3699	5194	203
node_2	20	2857	6681	235
node_3	12	3927	5014	208
node_4	24	3608	2555	252
node_5	10	3074	7246	257
TOTAL	**76**	**17165**	**26690**	**1155**

Job 1	Job2	Job3	Matching Scores		
node_1	node_1	No Match	1	1	0
node_2	node_2	No Match	1	1	0
node_3	node_3	No Match	1	1	0
No Match	No Match	No Match	0	0	0
node_5	node_5	No Match	1	1	0
Total Matching Score for			**0.80**	**0.80**	**0.00**

Qualified Nodes for Job1 in Cluster5				
node_1	10	3699	5194	203
node_2	20	2857	6681	235
node_3	12	3927	5014	208
node_5	10	3074	7246	257

Cluster 5 =			

Cluster 6

C6Node ID	NP	PS	MS	BW
node_1	20	4341	5531	196
node_2	10	5235	3052	173
node_3	14	8737	3694	170
node_4	9	8965	6326	280
node_5	45	3661	5335	200
node_6	7	4728	5627	200
node_7	20	5021	7349	260
node_8	10	3488	4352	161
node_9	9	6843	5612	128
node_10	22	3466	3268	156
TOTAL	**166**	**54485**	**50146**	**1924**

Job 1	Job2	Job3	Matching Scores		
node_1	node_1	No Match	1	1	0
node_2	node_2	No Match	1	1	0
node_3	node_3	No Match	1	1	0
node_4	node_4	No Match	1	1	0
node_5	node_5	No Match	1	1	0
node_6	node_6	No Match	1	1	0
node_7	node_7	No Match	1	1	0
node_8	node_8	No Match	1	1	0
node_9	node_9	No Match	1	1	0
node_10	node_10	No Match	1	1	0

Total Matching Score for Cluster 6 =	**1.00**	**1.00**	**0.00**

Qualified Nodes for Job1 in Cluster6				
node_1	20	4341	5531	196
node_2	10	5235	3052	173
node_3	14	8737	3694	170
node_4	9	8965	6326	280
node_5	45	3661	5335	200
node_6	7	4728	5627	200
node_7	20	5021	7349	260
node_8	10	3488	4352	161
node_9	9	6843	5612	128
node_10	22	3466	3268	156

Cluster 7

C7Node ID	NP	PS	MS	BW
node_1	22	5030	9851	109
node_2	3	6031	7481	108
node_3	10	5806	15400	151
node_4	7	5909	15282	137
node_5	4	5411	6722	176
node_6	2	5662	6537	171
node_7	30	5496	4609	159
node_8	8	5108	13368	189
node_9	14	3838	3423	113
node_10	9	3803	11185	131
TOTAL	**109**	**52094**	**93858**	**1444**

Job 1	Job2	Job3	Matching Scores		
node_1	node_1	No Match	1	1	0
node_2	node_2	No Match	1	1	0
node_3	node_3	No Match	1	1	0
node_4	node_4	No Match	1	1	0
node_5	node_5	No Match	1	1	0
node_6	node_6	No Match	1	1	0
node_7	node_7	No Match	1	1	0
node_8	node_8	No Match	1	1	0
node_9	node_9	No Match	1	1	0
node_10	node_10	No Match	1	1	0
Total Matching Score for Cluster 7 =			**1.00**	**1.00**	**0.00**

Qualified Nodes for Job1 in Cluster7				
node_1	22	5030	9851	109
node_2	3	6031	7481	108
node_3	10	5806	15400	151
node_4	7	5909	15282	137
node_5	4	5411	6722	176
node_6	2	5662	6537	171
node_7	30	5496	4609	159
node_8	8	5108	13368	189
node_9	14	3838	3423	113
node_10	9	3803	11185	131

Cluster 8

C8Node ID	NP	PS	MS	BW
node_1	4	5081	12895	211
node_2	4	7661	10093	170
node_3	6	3027	1545	211
node_4	3	2461	6441	163
node_5	10	5886	7715	223
node_6	2	2751	9280	203
node_7	6	5478	6896	185
node_8	6	5752	10048	140
node_9	5	5871	10969	226
node_10	6	4398	12064	200
TOTAL	**52**	**48366**	**87946**	**1932**

Job 1	Job2	Job3	Matching Scores		
node_1	node_1	No Match	1	1	0
node_2	node_2	No Match	1	1	0
No Match	No Match	No Match	0	0	0
No Match	No Match	No Match	0	0	0
node_5	node_5	No Match	1	1	0
node_6	node_6	No Match	1	1	0
node_7	node_7	No Match	1	1	0
node_8	node_8	No Match	1	1	0
node_9	node_9	No Match	1	1	0
node_10	node_10	No Match	1	1	0

Total Matching Score for Cluster 8 =	**0.80**	**0.80**	**0.00**

Qualified Nodes for Job1 in Cluster8				
node_1	4	5081	12895	211
node_2	4	7661	10093	170
node_5	10	5886	7715	223
node_6	2	2751	9280	203
node_7	6	5478	6896	185
node_8	6	5752	10048	140
node_9	5	5871	10969	226
node_10	6	4398	12064	200

Cluster 9

C9Node ID	NP	PS	MS	BW
node_1	9	5127	1619	236
node_2	4	6067	7270	267
node_3	7	4842	5085	283
node_4	10	3770	2500	207
node_5	8	5596	2955	227
node_6	10	5748	6258	255
node_7	6	2794	2125	256
TOTAL	**54**	**33944**	**27812**	**1731**

Job 1	Job2	Job3	Matching Scores		
No Match	No Match	No Match	0	0	0
node_2	node_2	No Match	1	1	0
node_3	node_3	No Match	1	1	0
No Match	No Match	No Match	0	0	0
node_5	node_5	No Match	1	1	0
node_6	node_6	No Match	1	1	0
No Match	No Match	No Match	0	0	0
Total Matching Score for Cluster 9 =			**0.57**	**0.57**	**0.00**

Qualified Nodes for Job1 in Cluster9				
node_2	4	6067	7270	267
node_3	7	4842	5085	283
node_5	8	5596	2955	227
node_6	10	5748	6258	255

Cluster 10

C10Node ID	NP	PS	MS	BW
node_1	10	3699	5194	203
node_2	28	2857	6681	235
node_3	14	3927	5014	208
node_4	8	3608	2555	252
node_5	10	3074	7246	257
TOTAL	70	17165	26690	1155

Job 1	Job2	Job3	Matching Scores		
node_1	node_1	No Match	1	1	0
node_2	node_2	No Match	1	1	0
node_3	node_3	No Match	1	1	0
No Match	No Match	No Match	0	0	0
node_5	node_5	No Match	1	1	0
Total Matching Score for			**0.80**	**0.80**	**0.00**

Qualified Nodes for Job1 in Cluster10				
node_1	10	3699	5194	203
node_2	28	2857	6681	235
node_3	14	3927	5014	208
node_5	10	3074	7246	257

Cluster 10 =	

Cluster 11

C11Node ID	NP	PS	MS	BW
node_1	7	4341	5531	196
node_2	8	5235	3052	173
node_3	22	2737	3694	170
node_4	9	3965	6326	140
node_5	16	3661	5335	114
node_6	7	4728	5627	105
node_7	30	5021	7349	169
node_8	10	2488	2352	161
node_9	9	2843	5612	128
node_10	12	3466	3268	156
TOTAL	**130**	**38485**	**48146**	**1512**

Job 1	Job2	Job3	Matching Scores		
node_1	node_1	No Match	1	1	0
node_2	node_2	No Match	1	1	0
node_3	node_3	No Match	1	1	0
node_4	node_4	No Match	1	1	0
node_5	node_5	No Match	1	1	0
node_6	node_6	No Match	1	1	0
node_7	node_7	No Match	1	1	0
No Match	No Match	No Match	0	0	0
node_9	node_9	No Match	1	1	0
node_10	node_10	No Match	1	1	0

Total Matching Score for Cluster 1 1=	**0.90**	**0.90**	**0.00**

Qualified Nodes for Job1 in Cluster11				
node_1	7	4341	5531	196
node_2	8	5235	3052	173
node_3	22	2737	3694	170
node_4	9	3965	6326	140
node_5	16	3661	5335	114
node_6	7	4728	5627	105
node_7	30	5021	7349	169
node_9	9	2843	5612	128
node_10	12	3466	3268	156

Cluster 12

C12Node ID	NP	PS	MS	BW
node_1	8	5030	9851	109
node_2	3	6031	7481	108
node_3	10	5806	15400	180
node_4	7	5909	15282	137
node_5	4	5411	6722	176
node_6	6	5662	6537	200
node_7	18	5496	4609	159
node_8	8	5108	13368	189
node_9	22	3838	3423	113
node_10	9	3803	11185	200
TOTAL	**95**	**52094**	**93858**	**1571**

Job 1	Job2	Job3	Matching Scores		
node_1	node_1	No Match	1	1	0
node_2	node_2	No Match	1	1	0
node_3	node_3	No Match	1	1	0
node_4	node_4	No Match	1	1	0
node_5	node_5	No Match	1	1	0
node_6	node_6	No Match	1	1	0
node_7	node_7	No Match	1	1	0
node_8	node_8	No Match	1	1	0
node_9	node_9	No Match	1	1	0
node_10	node_10	No Match	1	1	0
Total Matching Score for Cluster 12 =			**1.00**	**1.00**	**0.00**

Qualified Nodes for Job1 in Cluster12				
node_1	8	5030	9851	109
node_2	3	6031	7481	108
node_3	10	5806	15400	180
node_4	7	5909	15282	137
node_5	4	5411	6722	176
node_6	6	5662	6537	200
node_7	18	5496	4609	159
node_8	8	5108	13368	189
node_9	22	3838	3423	113
node_10	9	3803	11185	200

Stage 2: Computation of Node Performance Ration and Ranks

CAPCV = Cluster Aggregate Performance Value by configuration

Second filtering stage of selected resources based on computation of node performance evaluation is as shown the following cluster tables (Cluster 1 - 12).

	CLUSTER 1					
C1Node ID	**NP**	**PS**	**MS**	**BW**	**PF**	**Rank**
node_1	4	5030	9851	200	0.1198	3
node_2	10	6832	4052	300	0.1751	1
node_3	4	4341	3694	170	0.0968	6
node_4	9	4341	6326	140	0.1273	2
node_5	5	3661	5335	114	0.0916	8
node_6	4	4728	5627	105	0.0957	7
node_7	5	5021	7349	169	0.1165	4
node_8	0	0	0	0	NA	NA
node_9	9	2843	5612	128	0.1094	5
node_10	1	3466	3268	156	0.0678	9
TOTAL	**51**	**40263**	**51114**	**1482**		
					CAPVC =	**17507.1000**

	CLUSTER 2					
C2Node ID	**NP**	**PS**	**MS**	**BW**	**PF**	**Rank**
node_1	4	5030	9851	200	0.0884	7
node_2	6	6031	7481	250	0.1078	4
node_3	10	5806	15400	151	0.1210	1
node_4	10	5909	15282	137	0.1201	2
node_5	4	5411	6722	176	0.0854	9
node_6	4	5662	6537	171	0.0866	8
node_7	6	5496	4609	300	0.1061	5
node_8	8	5108	13368	189	0.1090	3
node_9	8	3838	3423	120	0.0811	10
node_10	9	3803	11185	131	0.0946	6
TOTAL	**69**	**52094**	**93858**	**1825**		
					CAPVC =	**25406.6000**

	CLUSTER 3					
C3Node ID	**NP**	**PS**	**MS**	**BW**	**PF**	**Rank**
node_1	10	5081	12895	211	0.1267	4
node_2	24	7661	10093	170	0.1933	1
node_3	0	0	0	0	NA	NA
node_4	0	0	0	0	NA	NA
node_5	10	5886	7715	223	0.1293	3
node_6	6	2751	9280	203	0.0848	8
node_7	6	5478	6896	185	0.1050	6
node_8	6	5752	10048	195	0.1128	5
node_9	14	5871	10969	226	0.1482	2
node_10	6	4398	12064	172	0.0998	7
TOTAL	**82**	**42878**	**79960**	**1585**		
					CAPVC =	**21209.2000**

	CLUSTER 4					
C4Node ID	**NP**	**PS**	**MS**	**BW**	**PF**	**Rank**
node_1	0	0	0	0	NA	NA
node_2	4	6067	7270	267	0.2359	4
node_3	7	4842	5085	283	0.2379	3
node_4	0	0	0	0	NA	NA
node_5	8	5596	2955	227	0.2410	2
node_6	10	5748	6258	255	0.2852	1
node_7	0	0	0	0	NA	NA
TOTAL	**29**	**22253**	**21568**	**1032**		
					CAPVC =	**9050.7000**

	CLUSTER5					
C5Node ID	**NP**	**PS**	**MS**	**BW**	**PF**	**Rank**
node_1	10	3699	5194	203	0.2333	4
node_2	20	2857	6681	235	0.2794	1
node_3	12	3927	5014	208	0.2519	2
node_4	0	0	0	0	NA	NA
node_5	10	3074	7246	257	0.2353	3
TOTAL	**52**	**13557**	**24135**	**903**		
					CAPVC =	**6682.0000**

	CLUSTER 6					
C6Node ID	**NP**	**PS**	**MS**	**BW**	**PF**	**Rank**
node_1	20	4341	5531	196	0.0994	5
node_2	10	5235	3052	173	0.0806	8
node_3	14	8737	3694	170	0.1145	4
node_4	9	8965	6326	280	0.1238	2
node_5	45	3661	5335	200	0.1396	1
node_6	7	4728	5627	200	0.0794	9
node_7	20	5021	7349	260	0.1147	3
node_8	10	3488	4352	161	0.0691	10
node_9	9	6843	5612	128	0.0910	6
node_10	22	3466	3268	156	0.0879	7
TOTAL	**166**	**54485**	**50146**	**1924**		
					CAPVC =	**21811.3000**

	CLUSTER 7					
C7Node ID	**NP**	**PS**	**MS**	**BW**	**PF**	**Rank**
node_1	22	5030	9851	109	0.1248	2
node_2	3	6031	7481	108	0.0775	10
node_3	10	5806	15400	151	0.1094	3
node_4	7	5909	15282	137	0.0999	5
node_5	4	5411	6722	176	0.0841	7
node_6	2	5662	6537	171	0.0796	9
node_7	30	5496	4609	159	0.1517	1
node_8	8	5108	13368	189	0.1017	4
node_9	14	3838	3423	113	0.0873	6
node_10	9	3803	11185	131	0.0840	8
TOTAL	**109**	**52094**	**93858**	**1444**		
					CAPVC =	**16899.1800**

	CLUSTER 8					
C8Node ID	**NP**	**PS**	**MS**	**BW**	**PF**	**Rank**
node_1	4	5081	12895	211	0.1185	7
node_2	4	7661	10093	170	0.1338	2
node_3	0	0	0	0	NA	NA
node_4	0	0	0	0	NA	NA
node_5	10	5886	7715	223	0.1630	1
node_6	2	2751	9280	203	0.0773	8
node_7	6	5478	6896	185	0.1253	5
node_8	6	5752	10048	140	0.1261	4
node_9	5	5871	10969	226	0.1324	3
node_10	6	4398	12064	200	0.1236	6
TOTAL	**43**	**42878**	**79960**	**1558**		
					CAPVC =	**21188.2000**

	CLUSTER 9					
C9Node ID	**NP**	**PS**	**MS**	**BW**	**PF**	**Rank**
node_1	0	0	0	0	NA	NA
node_2	4	6067	7270	267	0.2359	4
node_3	7	4842	5085	283	0.2379	3
node_4	0	0	0	0	NA	NA
node_5	8	5596	2955	227	0.2410	2
node_6	10	5748	6258	255	0.2852	1
node_7	0	0	0	0	NA	NA
TOTAL	**29**	**22253**	**21568**	**1032**		
					CAPVC =	**9050.7000**

	CLUSTER 10					
C10Node ID	**NP**	**PS**	**MS**	**BW**	**PF**	**Rank**
node_1	10	3699	5194	203	0.2240	4
node_2	28	2857	6681	235	0.2995	1
node_3	14	3927	5014	208	0.2505	2
node_4	0	0	0	0	NA	NA
node_5	10	3074	7246	257	0.2260	3
TOTAL	**62**	**13557**	**24135**	**903**		
					CAPVC =	**6686.0000**

	CLUSTER 11					
C11Node ID	**NP**	**PS**	**MS**	**BW**	**PF**	**Rank**
node_1	7	4341	5531	196	0.1068	5
node_2	8	5235	3052	173	0.1104	3
node_3	22	2737	3694	170	0.1186	2
node_4	9	3965	6326	140	0.1011	6
node_5	16	3661	5335	114	0.1092	4
node_6	7	4728	5627	105	0.0979	8
node_7	30	5021	7349	169	0.1719	1
node_8	0	0	0	0	NA	NA
node_9	9	2843	5612	128	0.0853	9
node_10	12	3466	3268	156	0.0987	7
TOTAL	**120**	**35997**	**45794**	**1351**		
					CAPVC =	**15696.7000**

	CLUSTER 12					
C12Node ID	**NP**	**PS**	**MS**	**BW**	**PF**	**Rank**
node_1	8	5030	9851	109	0.0883	8
node_2	3	6031	7481	108	0.0775	10
node_3	10	5806	15400	180	0.1155	3
node_4	7	5909	15282	137	0.1012	5
node_5	4	5411	6722	176	0.0837	9
node_6	6	5662	6537	200	0.0948	7
node_7	18	5496	4609	159	0.1242	1
node_8	8	5108	13368	189	0.1028	4
node_9	22	3838	3423	113	0.1170	2
node_10	9	3803	11185	200	0.0950	6
TOTAL	**95**	**52094**	**93858**	**1571**		
					CAPVC =	**25366.2000**

Stage 3: Selection of the Best Ranked Cluster

CLUSTER 2				
C2Node ID	NP	PS	MS	BW
node_1	4	5030	9851	200
node_2	6	6031	7481	250
node_3	10	5806	15400	151
node_4	10	5909	15282	137
node_5	4	5411	6722	176
node_6	4	5662	6537	171
node_7	6	5496	4609	300
node_8	8	5108	13368	189
node_9	8	3838	3423	120
node_10	9	3803	11185	131
TOTAL	69	52094	93858	1825

Appendix B: Job Description Example

```
0     user_33    q2    1     1     [p2;p10;p25]     1009884
      1228910204 1228910269 1231502311 2592042    265   812
1     user_33    q2    1     1     [p2;p10;p25]     1010132
      1228910204 1228910275 1231502321 2592046    265   813
2     user_33    q2    1     1     [p2;p10;p25]     1010108
      1228910223 1229948564 1231841932 1893368    271   807
3     user_33    q2    1     1     [p2;p10;p25]     2172  1228910223
      1229948565 1231774541 1825976    -4    809
4     user_44    q1    2     1     [p15] 4940  1229426624 1229426639
      1232018669 2592030    265   705 706
5     user_43    q1    1     1     [p19;p4;p15]     14088 1230194569
      1230194595 1232207557 2012962    0     673
6     user_43    q1    1     1     [p19;p4;p15]     13556 1230194569
      1230194595 1232484808 2290213    0     671
7     user_43    q1    1     1     [p19;p4;p15]     14124 1230194569
      1230194597 1232253522 2058925    0     672
8     user_43    q1    1     1     [p19;p4;p15]     14584 1230194570
      1230194596 1231977061 1782465    0     684
9     user_43    q1    1     1     [p19;p4;p15]     14112 1230194570
      1230194597 1232026296 1831699    0     674
10    user_8     q1    1     1     [p4;p15]   351604     1230299471
      1230299822 1231872830 1573008    1     695
11    user_8     q1    1     1     [p4;p15]   351028     1230299501
      1230299833 1231872830 1572997    1     696
12    user_8     q1    1     1     [p4;p15]   351504     1230387996
      1230388002 1231872830 1484828    1     687
13    user_8     q1    1     1     [p4;p15]   352136     1230390265
      1230390288 1231872831 1482543    1     688
14    user_9     q2    1     1     [p2;p10;p25]     1690068
      1230576438 1230576464 1230889074 312610     271   833
15    user_9     q2    1     1     [p2;p10;p25]     1558900
      1230580104 1230580113 1230842660 262547     271   837
16    user_8     q1    1     1     [p4;p15]   351792     1230644903
      1230644911 1231872835 1227924    1     682
17    user_1     q6    8     1     [p1;p2;p6;p7]    2989520
      1230675195 1230675200 1230768010 92810 271   16  17  18  19  20
      21 22 23
18    user_0     q3    4     1     [p32;p8]   6984  1230716510
      1230716517 1230801175 84658 0     593 594 595 596
19    user_0     q3    4     1     [p32;p8]   6992  1230716631
      1230716647 1230801665 85018 0     597 598 599 600
20    user_0     q3    4     1     [p32;p8]   6972  1230716750
      1230716756 1230801476 84720 0     605 606 607 608
```

The workload log was graciously provided by the Czech National Grid Infrastructure MetaCentrum (http://meta.cesnet.cz/).

Appendix C: Machine Description Example (metacentrum.mwf)

```
0     cluster_0  1500  48000000   Itanium2   linux
      p1,p2,p3,p4,p5,p6,p7,p8,p9 1     8     8
      16,17,18,19,20,21,22,23
1     cluster_1  2200  32000000   Opteron    linux
      p10,p11,p12,p3,p4,p13,p14,p15,p16,p9,p8     1     16    16
      24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39
2     cluster_2  3200  1009000    Xeon  linux
      p2,p17,p18,p4,p19,p14,p20,p15,p7,p8,p910    10
      1,1,1,1,1,1,1,1,1,1   42,43,44,45,46,47,48,49,50,51
3     cluster_3  2600  131182840  Opteron    linux
      p10,p11,p12,p4,p13,p14,p18,p21,p22,p23,p7,p16,p9,p8    5
      80    16,16,16,16,16
      76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,9
      5,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,
      111,112,113,114,115,116,117,118,119,120,121,122,123,124,12
      5,126,127,128,129,130,131,132,133,134,135,136,137,138,139,
      140,141,142,143,144,145,146,147,148,149,150,151,152,153,15
      4,155
4     cluster_4  1600  1005000    AthlonMP   linux
      p2,p4,p24,p25,p17,p18,p14,p26,p8,p9   16    32
      2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
      158,159,160,161,162,163,164,165,166,167,168,169,170,171,17
      2,173,174,175,176,177,178,179,180,181,182,183,184,185,186,
      187,188,189
5     cluster_5  2400  1048576    Xeon  linux
      p11,p19,p17,p18,p4,p27,p28,p14,p26,p21,p22,p7,p29,p8,p9,p30
      32    64
      2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
      2,2,2
      453,454,455,456,457,458,459,460,461,462,463,464,465,466,46
      7,468,469,470,471,472,473,474,475,476,477,478,479,480,481,
      482,483,484,485,486,487,488,489,490,491,492,493,494,495,49
      6,497,498,499,500,501,502,503,504,505,506,507,508,509,510,
      511,512,513,514,515,516
6     cluster_6  2659  15565060   Xeon  linux
      p11,p31,p4,p32,p13,p15,p14,p19,p6,p18,p21,p22,p23,p9,p8,p7
      36    148
      4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
      4,4,4,4,4,4,8
      517,518,519,520,521,522,523,524,525,526,527,528,529,530,53
      1,532,533,534,535,536,537,538,539,540,541,542,543,544,545,
      546,547,548,549,550,551,552,553,554,555,556,557,558,559,56
      0,561,562,563,564,565,566,567,568,569,570,571,572,573,574,
      575,576,577,578,579,580,581,582,583,584,585,586,587,588,58
      9,590,591,592,593,594,595,596,597,598,599,600,601,602,603,
      604,605,606,607,608,609,610,611,612,613,614,615,616,617,61
      8,619,620,621,622,623,624,625,626,627,628,629,630,631,632,
      633,634,635,636,637,638,639,640,641,642,643,644,645,646,64
```

```
     7,648,649,650,651,652,653,654,655,656,657,658,659,660,661,
     662,663,664
7    cluster_7  3056  2021000    Xeon  linux
     p33,p4,p19,p17,p18,p14,p26,p15,p8,p9,p7     35    70
     2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
     2,2,2,2,2,2,2
     665,666,667,668,669,670,671,672,673,674,675,676,677,678,67
     9,680,681,682,683,684,685,686,687,688,689,690,691,692,693,
     694,695,696,697,698,699,700,701,702,703,704,705,706,707,70
     8,709,710,711,712,713,714,715,716,717,718,719,720,721,722,
     723,724,725,726,727,728,729,730,731,732,733,734
8    cluster_8  1600  1024000    Opteron    linux
     p2,p4,p12,p25,p10,p13,p14,p26,p21,p22,p8,p9 10    20
     2,2,2,2,2,2,2,2,2,2
     807,808,809,810,811,812,813,814,815,816,817,818,819,820,82
     1,822,823,824,825,826
9    cluster_9  2400  4000000    Opteron    linux
     p2,p12,p25,p10,p13,p14,p21,p26,p22,p34,p4,p8,p9  3     6
     2,2,2 827,828,829,830,831,832
10   cluster_10 2000  4000000    Opteron    linux
     p2,p4,p12,p25,p10,p3,p23,p13,p14,p35,p21,p22,p34,p8,p9 23
     92    4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4
     833,834,835,836,837,838,839,840,841,842,843,844,845,846,84
     7,848,849,850,851,852,853,854,855,856,857,858,859,860,861,
     862,863,864,865,866,867,868,869,870,871,872,873,874,875,87
     6,877,878,879,880,881,882,883,884,885,886,887,888,889,890,
     891,892,893,894,895,896,897,898,899,900,901,902,903,904,90
     5,906,907,908,909,910,911,912,913,914,915,916,917,918,919,
     920,921,922,923,924
11   cluster_11 3000  4556800    Xeon  linux
     p2,p4,p32,p13,p14,p19,p6,p21,p36,p34,p8,p9,p15,p22     19
     152   8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8
     1023,1024,1025,1026,1027,1028,1029,1030,1031,1032,1033,103
     4,1035,1036,1037,1038,1039,1040,1041,1042,1043,1044,1045,1
     046,1047,1048,1049,1050,1051,1052,1053,1054,1055,1056,1057
     ,1058,1059,1060,1061,1062,1063,1064,1065,1066,1067,1068,10
     69,1070,1071,1072,1073,1074,1075,1076,1077,1078,1079,1080,
     1081,1082,1083,1084,1085,1086,1087,1088,1089,1090,1091,109
     2,1093,1094,1095,1096,1097,1098,1099,1100,1101,1102,1103,1
     104,1105,1106,1107,1108,1109,1110,1111,1112,1113,1114,1115
     ,1116,1117,1118,1119,1120,1121,1122,1123,1124,1125,1126,11
     27,1128,1129,1130,1131,1132,1133,1134,1135,1136,1137,1138,
     1139,1140,1141,1142,1143,1144,1145,1146,1147,1148,1149,115
     0,1151,1152,1153,1154,1155,1156,1157,1158,1159,1160,1161,1
     162,1163,1164,1165,1166,1167,1168,1169,1170,1171,1172,1173
     ,1174
12   cluster_12 2660  27343000   Xeon  linux
     p37,p4,p32,p13,p14,p19,p6,p18,p21,p36,p11,p38,p9,p8,p22     8
     64    8,8,8,8,8,8,8,8
     1175,1176,1177,1178,1179,1180,1181,1182,1183,1184,1185,118
     6,1187,1188,1189,1190,1191,1192,1193,1194,1195,1196,1197,1
```

```
      198,1199,1200,1201,1202,1203,1204,1205,1206,1207,1208,1209
      ,1210,1211,1212,1213,1214,1215,1216,1217,1218,1219,1220,12
      21,1222,1223,1224,1225,1226,1227,1228,1229,1230,1231,1232,
      1233,1234,1235,1236,1237,1238
13    cluster_13 2330  15200000   Xeon  linux
      p39,p4,p32,p13,p14,p19,p35,p18,p21,p36,p40,p9,p8,p22   11
      44    4,4,4,4,4,4,4,4,4,4,4,4
      1239,1240,1241,1242,1243,1244,1245,1246,1247,1248,1249,125
      0,1251,1252,1253,1254,1255,1256,1257,1258,1259,1260,1261,1
      262,1263,1264,1265,1266,1267,1268,1269,1270,1271,1272,1273
      ,1274,1275,1276,1277,1278,1279,1280,1281,1282
```

The workload log was graciously provided by the Czech National Grid Infrastructure MetaCentrum (http://meta.cesnet.cz/).

Appendix D: 2^K Factorial Design (k = 2)

Example:

This example presents a study of the impact of memory size and number of processor on the performance of a computational cluster. Our intent here is to determine certain resource configuration specifications responsible for selecting cluster resources appropriate for a particular problem run based on that run's characteristics. In this simple example, we considered both single and combined effects of two resource characteristics, number of processor per node and the memory attached to the processor on resource prioritization. Each resource characteristic is referred to as a factor. A resource can have several factors ranging from number of processor, processor speed, memory size, storage capacity, cache size, network bandwidth, etc. A factor is usually classified into two levels of priorities high (+) and low (-) as mentioned earlier in section 3.

Table I: Factors and levels for the initial two-level cluster resource configuration experiment

Factors		Memory Size (factor A)	
		4MB	16MB
Number of processor (factor B)	Level 1	14	46
	Level 2	25	75

i. memory size, two level

ii. number of processor, two level

Table II: Design and data for the initial two-level experiment: A 2^{2-1} design

		Memory Size			
		4MB		16MB	
Number of processor	1	-1	-1	1	-1
	2	-1	1	1	1

Representing the performance of a cluster as a regression model:

$$y = q_0 + q_A x_A + q_B x_B + q_{AB} x_A x_B$$

$$x_A = \begin{cases} -1 & if\ 4MB\ memory \\ 1 & if\ 16MB\ of\ memeory \end{cases} \quad and$$

$$x_B = \begin{cases} -1 & if\ 1\ processor \\ 1 & if\ 2\ processor \end{cases}$$

Computing the data, we have

Substituting the results into the model:

$$y_1 = q_0 - q_A - q_B + q_{AB}$$
$$y_2 = q_0 + q_A - q_B - q_{AB}$$
$$y_3 = q_0 - q_A + q_B - q_{AB}$$
$$y_4 = q_0 + q_A + q_B + q_{AB}$$

Solving equations for q_i

$$q_0 = 1/4(y_1 + y_2 + y_3 + y_4)$$
$$q_A = 1/4(-y_1 + y_2 - y_3 + y_4)$$
$$q_B = 1/4(-y_1 - y_2 + y_3 + y_4)$$
$$q_{AB} = 1/4(y_1 - y_2 - y_3 + y_4)$$
$$y = 40 + 21x_A + 10x_B + 5x_Ax_B$$

Interpretation:

– The mean performance is 40 MIPS

– Effect of memory is ±21 MIPS

– Effect of cache is ±10 MIPS

– Interaction (extra bonus) for cache and memory combination

accounts for ±5 MIPS

The sign table method (in table III) is introduced here to compute the effects of the different factors, sign table simply contains the effects of all factors in question. See table II.

Table III: Sign table method to determine the effects of factors for $k = 2$

I	A	B	AB	y	
1	-1	-1	1	14	
1	1	-1	-1	46	
1	-1	1	-1	25	
1	1	1	1	75	
160	82	40	18	Total	
40	21	10	5	Total/4	} Result

Determining the importance of a factor:

a. Importance of a performance factor: Variation of Factor/ Total variation

b. calculating sample variance of y $=$

$$s_y^2 = \frac{\sum_{i=1}^{2^2}(y_i - \bar{y})}{2^2 - 1}$$

c. calculating total variation or sum of squares total (SST) of y:

$$y = SST = \sum_{I=1}^{2^2}(y_i - \bar{y})^2$$

SST can be transformed to $SST = 2^2 q_A^2 + 2^2 q_B^2 + 2^2 q_{AB}^2$

These three parts represent the portion of the total variation explained by the effects of $A, B,$ and the interaction AB.

Substituting results from sign table 3 into SST gives:

$=2^2(21^2 + 10^2 + 5^2)$

=78% + 18% + 4%

Table 4 suggest that the effects of resource component A (processor memory) and resource component B (processor speed) are the largest. The linear effect of resource factor A is the most significant with an estimate of four times the estimate of the next most significant resource component, B; showing that resource component A is very significant and important relative to the second resource components'. The main effect A is responsible for 78% variation. While the interaction between effect A and B accounts for just 4% variation. Overall, the two main effects contribute 96% of the sum of squares, the two-factors interactions contribute 4%. In this cluster performance experiment, the main effects dominate the system. Therefore, a node with this resource specification can be matched to problems (application or job) with need for high memory demand.