NEW FUZZY SCHEDULING ALGORITHM (NFSA) FOR REAL-TIME TASKS ON MULTIPROCESSOR SYSTEMS

BY

MATTHEW TUNDE OGEDENGBE

DEPARTMENT OF MATHEMATICS

AHMADU BELLO UNIVERSITY, ZARIA

NIGERIA.

SEPTEMBER, 2014.

NEW FUZZY SCHEDULING ALGORITHM (NFSA) FOR REAL-TIME TASKS ON MULTIPROCESSOR SYSTEMS

BY

Matthew Tunde OGEDENGBE, B.Sc (ABU) 2010
M.Sc/SCI/4534/11-12

A THESIS SUBMITTED TO THE SCHOOL OF POSTGRADUATE STUDIES,
AHMADU BELLO UNIVERSITY, ZARIA
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD
OF A MASTER DEGREE IN COMPUTER SCIENCE.

DEPARTMENT OF MATHEMATICS,
FACULTY OF SCIENCE
AHMADU BELLO UNIVERSITY, ZARIA,
NIGERIA

SEPTEMBER, 2014.

Declaration

I declare that the work in this thesis entitled "New Fuzzy Scheduling Algorithm (NFSA) For Real-Time Tasks on Multiprocessor Systems" has been carried out by me in the Department of Mathematics, under the supervision of Dr. S.E. Abdullahi and Prof. S.B. Junaidu. The information derived from the literature has been duly acknowledged in the text and a list of references provided. No part of this thesis was previously presented for another degree or diploma at this or any other Institution.

Matthew Tunde OGEDENGBE _____ _____

Name of Student Signature Date

Certification

This thesis entitled "NEW FUZZY SCHEDULING ALGORITHM (NFSA) FOR REAL-TIME TASKS ON MULTIPROCESSOR SYSTEMS" by Matthew Tunde OGEDENGBE meets the regulations governing the award of the degree of Master of Science of Ahmadu Bello University and is approved for its contribution to knowledge and literary presentation.

Dr. S.E Abdullahi _____ _____

Chairman Supervisory Committee (Signature) (Date)

Prof. S. B Junaidu _____ _____

Member Supervisory Committee (Signature) (Date)

Dr. B. Sani _____ _____

Head of Department (Signature) (Date)

Prof. Adebayo A. Joshua _____ _____

Dean, School of Postgraduate Studies (Signature) (Date)

Dedication

I dedicate this thesis to my beloved mother, Olayinka Ogedengbe.

Acknowledgement

I give all glory, honor and adoration to God Almighty for giving me the exceeding grace and divine speed for the completion of my master program.

My gratitude and appreciation goes to my major supervisor Dr. S. E. Abdullahi and my minor supervisor Prof. S.B. Junaidu for giving me attention out of their tight schedules and also for guidance, contributions and directives for the success of this work, I say big thanks to you Sir.

I must acknowledge the scholarly preparations given to me by my Masters Degree instructors: Prof. S. B Junaidu, Prof. O.S Adewale, Prof. Hamawah and Dr. S.E Abdullahi whom I owe a debt of gratitude and pray God guidance as their rewards.

I wouldn't have been where I am today without the great support I received from my parents Mr. Samuel Ogedengbe and Mrs. Yinka Ogedengbe and my siblings Ezekiel, Esther and Lizzy and my uncle Lakers may God keep you and prosper your ways.

I acknowledge my very best friends in the department, Sani ThankGod, Abdulrazak, Olaide without you guys the java codes and system development of the work wouldn't have been a reality. I also remembered my group five member Rosemary, Vivian, ThankGod, Solomon for the knowledge and time we shared together toward the success of this work. I say thank you. My entire M.Sc classmates; I say a big thank to you all.

Dr. T. Aboiyar the head of maths/stats/comp department, University of Agriculture, Makurdi for providing the benevolent environment for my study; I pray God to lift you higher to next level.

My unreserved and golden appreciation goes to my spiritual father Pastor O. Steven; whose prayers and words of courage has always been my strength, I pray we will all make heaven in Jesus name (Amen).

Abstract

This thesis is an extension of existing fuzzy scheduling algorithms, to schedule real-time tasks on multiprocessor systems. The existing fuzzy scheduling algorithm was extended in order to have a better performance at higher system load. The task scheduling parameters used in this research work are; arrival time, computation time, and deadline which are inputs to the fuzzy inference system. The outputs are the runtime priorities which are used to schedule tasks in a priority (ready) queue for execution on multiprocessor. The performance of the new fuzzy scheduling algorithm was compared with the existing fuzzy scheduling algorithm. Results show that the new fuzzy algorithm has a better performance than the existing algorithm at higher system load; this is as a result of minimum response time, turnaround time and lower number of tasks that missed their deadline.

Table of Contents

Declaration i
Certification ii
Dedication iii
Acknowledgement iv
Abstract v
Table of Contents vi
List of Tables ix
List of Figures x
List of Appendices xi
List of Abbreviations xii
CHAPTER ONE 1
GENERAL INTRODUCTION 1
1.0 INTRODUCTION 1
1.1 PROBLEM STATEMENT AND RESEARCH MOTIVATION 2
1.2 RESEARCH OBJECTIVES 3
1.3 RESEARCH METHODOLOGY 3
1.4 CONTRIBUTION OF THE STUDY TO KNOWLEDGE 3
1.5 ORGANIZATION OF THE THESIS 4
CHAPTER TWO 5
LITERATURE REVIEW 5
2.0 INTRODUCTION 5
2.1 REAL-TIME SYSTEMS 5
2.2 REAL-TIME SYSTEMS FEATURES 7
2.3 REAL-TIME SYSTEM SCHEDULING 7
2.4 REAL TIME SYSTEM SCHEDULING ALGORITHMS 8
2.5 MULTIPROCESSOR SCHEDULING 10
2.6 TASK UTILIZATION (LOAD) FACTOR 11
2.7 INTRODUCTION TO FUZZY LOGIC 12
2.7.1 Crisp Set Theory 13
2.7.2 Fuzzy Set Theory 13
2.7.3 Membership Functions 14

2.7.4 Linguistic Variables 15
2.7.5 Fuzzy Logic Systems (FLS) 16
2.8 RELATED WORKS 18
CHAPTER THREE....23
NEW FUZZY SCHEDULING ALGORITHM (NFSA) DESIGN....23
3.0 INTRODUCTION....23
3.1 NEW FUZZY SCHEDULING ALGORITHM (NFSA) MODEL 23
3.1.1 Defining the Inputs/Output Membership Function for NFSA....25
3.1.2 Set-up Fuzzy Rule Base for NFSA 28
3.1.3 Defuzzified the Output for NFSA....28
3.2 CASE ASSUMPTION 32
3.3 NFSA ARCHITECTURE....33
3.4 NFSA ALGORITHM....35
3.5 FLOW CHART FOR NFSA 37
CHAPTER FOUR....38
IMPLEMENTATION AND TEST OF THE NFSA....38
4.0 INTRODUCTION....38
4.1 ILLUSTRATIVE EXAMPLE 38
4.1.1 Performance Evaluation Metric of EFA....40
4.1.2 Performance Evaluation Metric of NFSA 43
4.1.3 Comparing NFSA with EFA 46
4.2 SYSTEM REQUIREMENT....46
4.3 USER INTERFACE....47
4.4 RESULTS DISCUSSION AND CONCLUSION 50
4.4.1 Graphical Representation of Average Turnaround Time. 50
4.4.2 Graphical Representation of Average Response Time. 52
4.4.3 Graphical Representation of Number of Tasks Miss Deadline 54
4.5 PERFORMANCE CRITERIA ANALYSIS TABLE 56
CHAPTER FIVE....58
SUMMARY, CONCLUSION AND FUTURE WORKS 58
5.1 SUMMARY....58
5.2 CONCLUSION....58
5.3 FUTURE WORKS....59

REFERENCES 60
APPENDICES 64

List of Tables

Table 2.1: Example of Task Utilization for 3 Tasks on Uniprocessor. 12
Table 3.1: Fuzzy Variable Range for Arrival Time 25
Table 3.2: Fuzzy Variable Range for Computation Time 26
Table 3.3: Fuzzy Variable Range for Deadline. 26
Table 3.4: Fuzzy Variable Range for Runtime Priority. 27
Table 3.5: Fuzzy Rules for NFSA 28
Table 4.1: Tasks Generated under Normal Load for Three (3) Processors. 39
Table 4.2: Output (Runtime Priority) of both EFA and NFSA 40
Table 4.3a: EFA Sorted Tasks 40
Table 4.3b: NFSA Sorted Tasks. 40
Table 4.4: EFA Deadline Missed 43
Table 4.5: NFSA Deadline Missed 46
Table 4.6: Comparing EFA and NFSA 46
Table 4.7a: Average Turnaround Time. 57
Table 4.7b: Average Response Time. 57
Table 4.7c: Total Number of Deadline Missed. 57

List of Figures

Figure 2.1: Tasks Attributes/Parameters 6
Figure 2.2: Triangular Membership Function 15
Figure 2.3: Fuzzy Computation Time Modelling (Author's source). 16
Figure 2.4: Fuzzy Logic System (FLS) 18
Figure 2.5: Existing Fuzzy Inference Model (Sheo *et al., 2012*) 22
Figure 3.1: New Fuzzy Inference Model. 24
Figure 3.2: Membership Function corresponding to Arrival Time. 25
Figure 3.3: Membership Function corresponding to Computation Time. 26
Figure 3.4: Membership Function corresponding to Deadline. 27
Figure 3.5: Membership Function corresponding to Runtime Priority. 27
Figure 3.6: Triangular Membership Function for High at 0.25 31
Figure 3.7: Triangular Membership Function for Normal at 0.25 31
Figure 3.8: Triangular Membership Function for High at 0.30 31
Figure 3.9: Triangular Membership Function for Normal at 0.75 32
Figure 3.10: Defuzzification with Centroid Approach 32
Figure 3.11: New Fuzzy Scheduler Architecture. 35
Figure 3.12: Flow Chart for NFSA 37
Figure 4.1: Gantt Chart EFA Tasks Assignment on Three (3) Processors 41
Figure 4.2: Gantt Chart NFSA Tasks Assignment on Three (3) Processors 44
Figure 4.3: NFSA Users Interface. 48
Figure 4.4: Results Analysis of EFA and NFSA after Simulation. 49
Figure 4.5: Average Turnaround Time on 3 Processors 51
Figure 4.6: Average Turnaround Time on 10 Processors 51
Figure 4.7: Average Turnaround Time on 100 Processors. 52
Figure 4.8: Average Response Time on 3 Processors 53
Figure 4.9: Average Response Time on 10 Processors 53
Figure 4.10: Average Response Time on 100 Processors. 54
Figure 4.11: Number of Tasks Missed Deadline on 3 Processors 55
Figure 4.12: Number of Tasks Missed Deadline on 10 Processors 55
Figure 4.13: Number of Tasks Missed Deadline on 100 Processors 56

List of Appendices

Appendix I: The 27 Fuzzy Rules of the proposed NFSA.....64
Appendix II: Results of the Simulation in Data.txt.....65
Appendix III: Java Source Codes for NFSA 67

List of Abbreviations

ABT: Average Burst Time

API: Application Programming Interface

ART: Average Response Time

AT: Arrival Time

ATAT: Average Turnaround Time

BT: Burst Time

CT: Computation Time

CW: Computation Weight

DLM: Deadline Missed

EDF: Earliest Deadline First

EFA: Existing Fuzzy Algorithm

Epriority: External Priority

FLS: Fuzzy Logic Systems

GUI: Graphical User Interface

HRRN: Highest Response Ratio Next

IDE: Integrated Development Environment

JDK: Java Development Kits

LLF: Least Laxity First

MF: Membership Function

NFSA: New Fuzzy Scheduling Algorithm

RAM: Random Access Memory

RM: Rate Monotonic

SJF: Shortest Job First

TAT: Turnaround Time

TQ: Time Quantum

WCET: Worst Case Execution Time

CHAPTER ONE

GENERAL INTRODUCTION

1.0 INTRODUCTION

Scheduling algorithm is more significant in a real-time system to ensure desired and predictable behaviour of a system (Sagar *et al*., 2012). Scheduling is concern with the optimal allocation of resources to tasks in a specified time. Real-time systems are often embedded in most of domestic devices which are used for daily activities without our knowledge when we use the devices in which they reside. Cars, planes and entertainment systems are just some devices in which real-time systems reside, governing the workings of that device while we do not consider that such a system exists within the chosen device. A real-time-system is a computer system in which the key aspect of the system is to perform tasks on time, not finishing too early nor too late. A classic example is that of the air-bag in a car; it is of great importance that the bag inflates neither too soon nor too late in orders to be of help and not be potentially harmful. In real-time systems, all tasks have specific parameters such as execution time, deadline, arrival time, priority, etc. Modern embedded computing systems are becoming increasingly complex (Sagar *et al*., 2012).

Many real-time systems are soft in which missing the deadline is tolerable while some others are hard and missing deadline is catastrophic (Sabeghi *et al.,* 2006). Nowadays, the use of real-time multiprocessor systems is dramatically increasing. Unfortunately, little is known about how to schedule multiprocessor-based real-time systems than that for uni-processors. The multiprocessor based scheduling have more computational complexity in practical algorithm which are unknown to most researchers, this open floor for new area of research in operating systems (Shahzad and Afzal, 2006). It has been provened that finding a minimal schedule for a set of real-time tasks in multiprocessor

system is a hard problem (Tanenbaum and Woodfhull, 2005). In this research fuzzy technique was introduced to schedule tasks on multiprocessors in real-time system. Many scheduling algorithms have been studied to guarantee the time constraints of real-time. Scheduling decision of real-time algorithms is usually based on parameters which are assumed to be crisp. However, in uncertain and imprecise environment the values of these parameters are vague. The vagueness of these parameters suggests that we make use of fuzzy logic to capture uncertainty and imprecision mode. Fuzzy scheduling approach therefore help to decide in what order the tasks should be scheduled for execution to better utilize the system and as a result reduce the chance of task' deadline being missed.

1.1 PROBLEM STATEMENT AND RESEARCH MOTIVATION

There are many conventional approaches to scheduling algorithms to guarantee the time constraints of real-time processes but no one is absolutely ideal. Scheduling decision of these algorithms is usually based on parameters which are assumed to be crisp. All practical real-time scheduling algorithms in multiprocessor systems present a trade-off between their computational complexity and performance. In real-time systems, tasks have to be performed correctly and timely. Finding minimal schedule in multiprocessor systems with real-time constraints is shown to a hard problem (Sheo *et al*., 2012). Although some optimal algorithms have been employed in uni-processor systems, they fail when they are applied in multiprocessor systems. However, the existing algorithm has a better performance at normal system load but at overloaded condition the performance decreases. Therefore, this research work will focus on the development and implementation of a new fuzzy algorithm for fair scheduling of real-time tasks on multiprocessor in order to improve performance of an existing system at higher system load.

1.2 RESEARCH OBJECTIVES

The specific objectives of this research are to:

i. Develop a new scheduling algorithm for real-time tasks on multiprocessor systems.
ii. Extend the existing fuzzy scheduler model using fuzzy logic techniques.
iii. Implement the new scheduling algorithm on the extended fuzzy scheduler.
iv. Evaluate performance metrics of both existing and new fuzzy scheduling algorithm on multiprocessor systems in term of turnaround time, response time and number of real-time tasks misses deadline at higher system load.

1.3 RESEARCH METHODOLOGY

This research applies fuzzy logic techniques to control the scheduling activities of multiprocessor in real-time environment by putting into consideration all major scheduling parameters of each task. Review of related literatures on real-time systems and fuzzy logic scheduling algorithm were carried out.

The fuzzy logic techniques bring an expert skill to solve problems with imprecise data or incomplete knowledge. In each test case, the number of tasks and some of the metric parameters will be randomly generated using Poisson, uniform distribution and normal distribution (Sheo *et al.*, 2012). Prototype software will be developed and implemented using java programming language and Netbeans editor.

1.4 CONTRIBUTION OF THE STUDY TO KNOWLEDGE

This research work uses fuzzy inference engine as a tool to model a knowledgeable decision making on how to schedule the processors to a given task in real-time environment. A fuzzy logic technique is adopted because of its capability in bringing an

expert's skill to solve problems of incomplete knowledge or imprecise data. This bring about;

i. An idea of using Fuzzy logic approach scheduling algorithm to optimized performance metrics of multiprocessor systems in real-time environment.
ii. A new concept of scheduling on multiprocessor systems.
iii. Carefully selecting scheduling parameters improves system performance with the appropriate algorithms.

1.5 ORGANIZATION OF THE THESIS

The rest of the thesis is organized as follows: the second chapter involves basic concept of real-time systems, multiprocessor scheduling, brief introduction of fuzzy logic and the review of related literature. Third chapter involves design of the new fuzzy scheduling algorithm. System implementation and simulation results obtained from the new fuzzy scheduling algorithm are discussed in the fourth chapter. Finally the fifth chapter involves summary, conclusion and future work.

CHAPTER TWO

LITERATURE REVIEW

2.0 INTRODUCTION

In modern world of computing real-time systems play an important role without which human daily tasks cannot be conveniently or successfully executed (Shatha and Kasim, 2010). Various household real-time devices are used on a daily bases but users know little or nothing about them. From mobile to missile, medical imaging systems, industrial control systems, display systems, Space Shuttle avionics system, traffic control, automated factory, military systems and various scientific experiments which require real-time communication and computation (Stallings, 2004).

2.1 REAL-TIME SYSTEMS

In real-time systems scheduling plays a more critical role than non-real-time systems because in these systems having the right answer too late is as bad as not having it at all (Ramamritham and Stankovic, 1994). Such a system must react to the requests within a fixed amount of time which is called deadline. In these systems processes are referred to as tasks and these have certain temporal qualities and restrictions (Cottet *et al.*, 2002). All real-time tasks consist of certain attributes or parameters such as; deadline, execution (or computation) time and release time. However, there are other temporal attributes that may be assigned to a task. The three mentioned are the basic ones. The release (or ready) time, is the required time at which real-time tasks are made ready for execution. The deadline is the maximum time required for a given task to have done with execution and the execution time is the required time take to run a given task on processor.

However, most tasks are recurring and have a period in which it executes. Such a task is referred to as periodic task. The period is the time interval from when a task may start until when the next instance of the same task may start again (Fredrik *et al.*, 2010). The figure 2.1 depicts the basic temporal attributes/parameter of a real-time task.

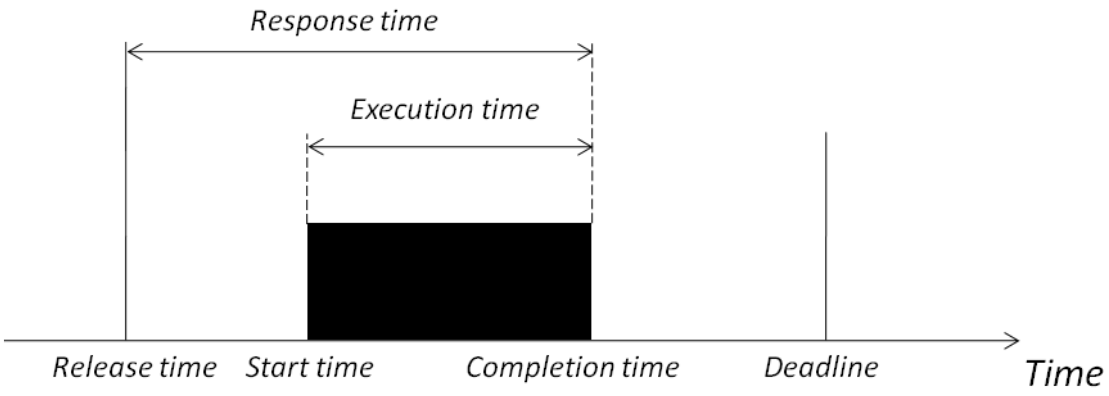


Figure 2.1: Tasks Attributes/Parameters

i. Release/ready time: The time a task is ready to run and just waits for the scheduler to activate it.
ii. Deadline: The maximum time a task must be done with execution.
iii. Execution/run time: The active computation time a tasks need to complete.
iv. Worst Case Execution Time (WCET): The longest possible execution time for a task on a particular type of system.
v. Response time: The time at which a task received first response for execution. Measured from release time to execution completes, including pre-emptions.
vi. Priority/weight: The importance given a task in context of the schedule at hand.

Moreover, real-time systems have requirements based on deadline which categorised them into two major groups; hard real-time system and soft real-time.

i. **Hard Real-time Systems:** is never allowed to miss a deadline because missing the deadline result to complete system failure. A hard real-time system can be safety-critical and this means that if a deadline is missed it can lead to catastrophic consequences which can harm persons or the environment.

ii. **Soft Real-time Systems:** In soft real-time systems, the system can miss some deadline. This will only result to degradation in the system performance.

2.2 REAL-TIME SYSTEMS FEATURES

A real-time system must have following features for the efficient working of the system:

i. Fast Response and Predictability: The systems should respond predictably fast to the urgent events so as their deadlines are met.

ii. High degree of schedulability: The timing requirements must be satisfied at high degrees of resource usage. The constraints should be met even if all the resources are being utilized i.e. high degree of schedulability should be achieved.

iii. Stability under transient overload: The system must be stable under overloaded conditions.

iv. Efficiency: As most real-time systems are embedded into small devices with space, weight, memory, power constraints, an efficient management of available resources is essential for achieving a desired performance.

v. Fault Tolerance: The system should not crash on the occurrence of single hardware or software failure (Robin, 2012).

2.3 REAL-TIME SYSTEM SCHEDULING

Real-Time Scheduling System consists of scheduler, clock and the processing hardware elements. In a real-time system, processes or tasks have schedulability; tasks are loaded

into real-time system and completed as specified by the tasks deadline depending on the characteristic of the scheduling algorithm for the task execution (Liu and Layland, 1973). There are two basic scheduling algorithms in real-time system and they can be categorised into offline scheduling algorithms and online scheduling algorithms.

i. **Offline/Static Scheduling:** In offline scheduling the scheduler have complete knowledge of all tasks and scheduling decisions to be taken before the system is started. During runtime the tasks are executed in a predetermined order. Offline scheduling is useful if we have a hard real-time system with complete knowledge of all the tasks because then a schedule of the tasks can be made which ensures that all tasks will meet their deadlines, if such a schedule exists.
ii. **Online/Dynamic Scheduling:** In online scheduling the decisions of how to schedule the tasks are known during execution time. The scheduling decisions are based on the tasks priorities which are either assigned dynamically or statically. Static priority driven algorithms assigned priorities to the tasks before the start of the system. Dynamic priority driven algorithms assign the priorities to tasks during runtime (Bovet and Marco, 2005).

2.4 REAL TIME SYSTEM SCHEDULING ALGORITHMS

For the purpose of this research work, three real-time algorithms were reviewed they are rate monotonic (RM), earliest deadline first (EDF) and least laxity first (LLF)

i. **Rate Monotonic (RM) algorithm:** is a uniprocessor static-priority preemptive scheme. The algorithm is static-priority in the sense that all priorities are determined for all instances of tasks before runtime (Strosnider *et al*., 1995). The priority of a task is been determined by the length of it period. Tasks with short period times are

assigned higher priority. RM is used to schedule periodic tasks. The following are preconditions for the rate monotonic algorithm (Liu and Layland, 1973).

a. Periodic tasks have constant known execution times and are ready for execution at the beginning of each period (T).
b. Deadlines (D) for tasks are at the end of each period: (D = T)
c. The tasks are independent, that is, there is no precedence between tasks and they do not block each other.
d. Scheduling overhead due to context switches and swapping are assumed to be zero.

ii. **Earliest Deadline First (EDF):** is a dynamic priority driven scheduling algorithm which gives tasks priority based on deadline. The preconditions for RM are also valid for EDF. The task with the currently earliest deadline during runtime is assigned the highest priority. That is if a task is executing with the highest priority and another task with an earlier deadline becomes ready it receives the highest priority and therefore preempts the currently running task and begins to execute. EDF is an optimal dynamic priority driven scheduling algorithm with preemption for a real-time system on a uniprocessor. EDF is capable of achieving full processor utilization (Liu and Layland, 1973).

iii. **Least laxity First (LLF):** is also known as least slack time, is a dynamic priority driven scheduling algorithm that assigns priority based on the laxity. The definition of laxity is the tasks deadline minus the remaining computation time of the task. It can also be described as the maximum time a task can wait before it needs to execute in order to meet its deadline. The task with the currently least laxity is assigned the

highest priority and is therefore executed. The executing task will be preempted by any other task with a currently smaller laxity. When the task is executing the laxity remains constant. If two tasks have similar laxity they will continually preempt each other and it will therefore create many context switches. But if we ignore the cost of the context switches LLF is also, as EDF, an optimal dynamic scheduling algorithm (Dertouzos and Mok, 1989).

2.5 MULTIPROCESSOR SCHEDULING

In multiprocessor scheduling, algorithms developed for uniprocessor scheduling can be applied if we consider each core of the multiprocessor as an isolated core, which is a uniprocessor. However, in multiprocessor scheduling the difficulty of verifying that the execution of different tasks on multiple cores does not interfere with each other and also determining which tasks should be given to a certain core increases the complexity greatly compared to uniprocessor scheduling (Bogdan *et al*., 2005). Moreover scheduling algorithms for multiprocessors often involves heuristics to simplify the task of finding a feasible schedule. The two main approaches for scheduling algorithms on multiprocessors are global scheduling and partitioning scheduling (Bovet and Marco, 2005).

i. **Global scheduling algorithms:** In these algorithms ready tasks are put in a global queue sorted based on priority. The task with currently highest priority, which is first in the queue, is selected by the scheduler and will execute on one of the processors and may be preempted or migrated to another processor if necessary.
ii. **Partitioning scheduling algorithms:** These algorithms assigned each task to one processor and will exclusively execute on that processor. In this approach each processor has its own local queue where tasks are been scheduled for execution. This approach exhibits uniprocessors in a multiple pattern (Carpenter *et al*., 2004).

2.6 TASK UTILIZATION (LOAD) FACTOR

Let a set *T* consists of *n* periodic tasks; the task/process utilization factor of a task T_i from *T*, denoted by U_i, is the fraction of time used in execution of a task. Since $\frac{C_i}{T_i}$ is the utilization factor of the task T_i, the total process utilization factor of the set is given by:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}, \tag{2.1}$$

where *U* is the utilization factor, C_i is the computation time, T_i is the release period (equal to deadline), and *n* is the number of tasks to be scheduled. The process utilization factor represents the load on a processor during the scheduling of periodic tasks.

However, it has been proved that the task utilization bound is

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n\left(2^{\frac{1}{n}} - 1\right). \tag{2.2}$$

This implies that when the load factor of tasks does not exceed their utilization bound, there is every tendency that all tasks will meet deadline but as soon as their utilization factor exceeds their utilization bound, the number of tasks meeting their deadline is not guaranteed. For instance, the utilization bound of two tasks on one processor is $U \leq 0.8284$ for $n = 2$, from equation 2.2. When the number of tasks tends towards infinity this expression will tend towards:

$$\lim_{n\to\infty} n\left(2^{1/n} - 1\right) = \ln 2 \approx 0.6931 \tag{2.3}$$

Therefore, in general most tasks will meet deadlines if CPU utilization is 69.3%. Also, it has been proved (Liu and Layland, 1973) that the system is fully utilized if $U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$, that is, 100% utilized on one processor. However, for two processors will be 200%, three processors will be 300% for full utilization and so on.

Table 2.1: Example of Task Utilization for 3 Tasks on Uniprocessor.

Process	Execution Time	Period
P1	1	8
P2	2	5
P3	2	10

From table 2.1, the processor utilization can be computed as follows:

$\frac{1}{8}+\frac{2}{5}+\frac{2}{10}=0.725$, which implies 72.5% of the processor was utilized

$U=3\left(2^{\frac{1}{3}}-1\right)=0.77976$, this mean that 77.976% is the utilization bound for three tasks on one processor.

Where U is called the utilization bound for three tasks. From the example above, the utilization bound for three (3) processes is 0.77976, which implies that at 77.976% of the processor all the tasks can meet their deadline. Since $0.725 < 0.77976$ the system is surely schedulable and the scheduling algorithm can meet all the deadlines (Liu and Layland, 1973).

2.7 INTRODUCTION TO FUZZY LOGIC

The concept of fuzzy sets was firstly introduced by Zadeh in 1965. The author's intention was to create a mathematical representation for handling data imprecision and information possessing non-statistical uncertainties. Basically, fuzzy sets are a supplement of ordinary crisp sets. To characterize differences between the two paradigms and capabilities of fuzzy sets, it is necessary to take a look at the definition of crisp sets and compares them to fuzzy set definitions.

2.7.1 Crisp Set Theory

In various possibilities to describe a crisp set, naming its member or stating a predicate, it is possible to use a *characteristic or indicator function* to distinguish between elements contained and not contained in a set. This function U assigns either 1 or 0 to elements i.e. $X_A: U \rightarrow \{0,1\}$

$$X_A(x) = \begin{cases} 1, & if\ x \in A \\ 0, & if\ x \notin A \end{cases} \tag{2.4}$$

The set A is then defined by the support of U which are all elements with U greater than zero.

2.7.2 Fuzzy Set Theory

Fuzzy set is a class of objects with a continuum of grades of membership. It generalizes the concept of a classical set, allowing its objects to have a degree of membership called *membership function* which assigns to each object ranging between zero and one. They relax the concept of *characteristic functions* by allowing assigning continuous values to objects of a particular universe (Zadeh, 1965). Due to this fact, it is possible to express partial affiliations to sets by computing a degree of membership rather than dichotomizing elements of a universe (Pedrycz and Gomide, 2007). The support of U denotes a set containing every element with a membership degree greater than zero. Most of the time, membership functions have common characteristic shapes. However, detailed membership functions employed in this research work will be presented in Chapter 3.

However, to devise a concise theory of Fuzzy logic, fuzzy logic can be defined as an extension of Boolean logic dealing with the concept of partial truth which denotes the extent to which a proposition is true (Sheo *et al*., 2012). Fuzzy logic provides an inference morphology that enables approximate human reasoning capabilities to be applied to knowledge-based systems. The theory of fuzzy logic provides a mathematical strength to

capture the uncertainties associated with human cognitive processes, such as thinking and reasoning. The development of fuzzy logic was motivated in large measure by the need for a conceptual framework which can address the issue of uncertainty and lexical imprecision. Some of the essential characteristics of fuzzy logic relate to the following:

i. In fuzzy logic, everything is a matter of degree.

ii. Any logical system can be fuzzified.

2.7.3 Membership Functions

A membership function (MF) for a fuzzy set A on the universe of discourse X is defined as a curve that defines how each point (element of X) in the input space is mapped to a membership value (or degree of membership) between 0 and 1. The input space is sometimes referred to as the *universe of discourse*. It can be mathematically denoted as

$$\mu_A : \mathrm{X} \rightarrow [0,1] \, . \tag{2.5}$$

This value μ_A, called membership value or degree of membership, quantifies the grade of membership of the element in X to the fuzzy set A. There are various types of MF been used in fuzzy logic they are; triangular MF, trapezoidal MF, Gaussian MF and so on. Among MFs listed, triangular MFs have been used extensively, especially in real-time implementations. Therefore this research work considered triangular MFs due to their simple formulas and computational efficiency as stated in equation 2.6.

$$\mu_A(x) = \begin{cases} 0, & x \leq a \\ \frac{x-a}{m-a}, & a < x \leq m \\ \frac{b-x}{b-m}, & m < x < b \\ 0, & x \geq b \end{cases} \tag{2.6}$$

A triangular MFs are defined by three parameters {a, m, b} where a$< m < b$. The x coordinates of the underlying triangular MF in figure 2.2 were determined by a, m and b. Membership functions allow us to graphically represent a fuzzy set. As shown in figure 2.2, the x axis represents the universe of discourse, whereas the y axis represents the degrees of membership in the [0, 1] interval (Bindi, 2013).

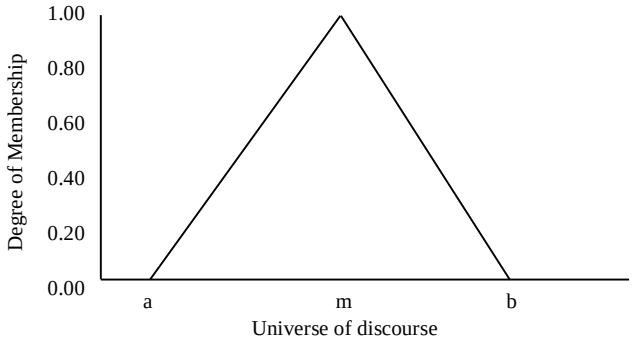


Figure 2.2: Triangular Membership Function

2.7.4 Linguistic Variables

Often, fuzzy sets describe measurements of physical phenomena, e. g. temperature, position, or size. Mathematically, those measurements are formalized using variables. In the sense of fuzzy set theory this concept was supplemented by linguistic variables. They are based on reference variables with defined ranges, for example, U = [0, 15]. However, they are further specified using linguistic terms, describing their meaning or purpose. Moreover, they contain multiple linguistic states which correspond to fuzzy sets. Hence, they are an analogy of family of sets for crisp sets. Figure 2.3 exemplifies the modelling of computation time. In this figure, a linguistic variable computation time bases on a reference variable U lying between [0, 15]. It uses three linguistic states with different fuzzy sets. As illustrated, the sets have no strict borders, instead depending on particular computation time the memberships increase or decrease. Using these variables it is

possible to formulate linguistic statements better understandable by human-beings. For instance, it is easier to understand that *computation time is long* is formulated with a linguistic variable and its states, than using numeric values the computation time is between 7.5 and 15.0ms.

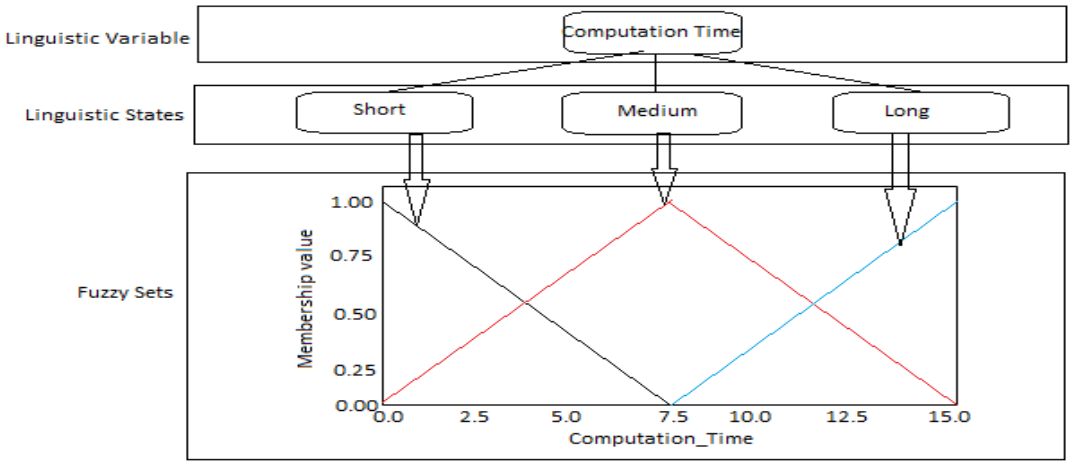


Figure 2.3: Fuzzy Computation Time Modelling (Author's source).

2.7.5 Fuzzy Logic Systems (FLS)

Most of the systems that are constructed based on fuzzy sets and logic has a common architecture. This FLS based on a specific lifecycle model consisting of four characteristic stages (William, 2011). As shown in Figure 2.4, the four components of a fuzzy logic system are explained as followed;

i. **Fuzzifier Component**: Here, the crisp input data which is captured from the environment is processed and interpreted using fuzzy sets, i.e. the appropriate linguistic variables states are identified and the degree of membership is determined. According to the computation time example shown in Figure 2.3, we would receive various computation time measured as input which would then be

represented using the three linguistic states with their fuzzy sets and memberships. After that, the resulting fuzzified values are delivered into the fuzzy inference engine.

ii. **Fuzzy Inference Engine:** As depicted in figure 2.5, Fuzzy Inference Engine is conceptually very simple. It consists of an input, a processing, and an output stage. The input stage maps the inputs, such as external priority (Epriority), arrival time, deadline, computation time and so on, to the appropriate membership functions and truth values. The processing stage invokes each appropriate rule and generates a corresponding result. It then combines the results. Finally, the output stage converts the combined result back into a specific output value (Thai, 2002). There are two common inference methods. First is called Mamdani's fuzzy inference method proposed in 1975 by Ebrahim Mamdani (Mamdani and Assilian, 1975) and the other is Takagi-Sugeno-Kang, or simply Sugeno, introduced in 1985. These two methods are the same in many respects, such as the procedure of fuzzifying the inputs and fuzzy operators. The main difference between Mamdani and Sugeno is that the Sugeno output membership functions are either linear or constant but Mamdani's inference expects the output membership functions to be fuzzy sets (Buttazzo *et al*., 2002).

iii. **Fuzzy Rules:** these concatenate linguistic variables together with their particular states and memberships. They use the predicates and logical operators.

Logical Operator: for standard logical operations to hold, fuzzy values only need to be held by their extreme boundaries of 1 and 0, the input values can be real numbers between 0 and 1. The other difference is the use of intersection, union,

and the complement operators such as "MIN(A,B), MAX(A,B) and 1-A" instead of the "A AND B, A OR B and NOT A". There are other operators other than the three stated above, but these are the most commonly used ones for science and engineering applications. However, more description will be giving in the next chapter.

iv. **Defuzzifier Component:** this is responsible for generating proper values depending on particular output variables. In this work only the centre of gravity method was applied for generating single values out of combined fuzzy sets. It computes the defuzzified result by determining the element for which the combined set's area is divided into two equal faces (Klir and Yuan, 1995).

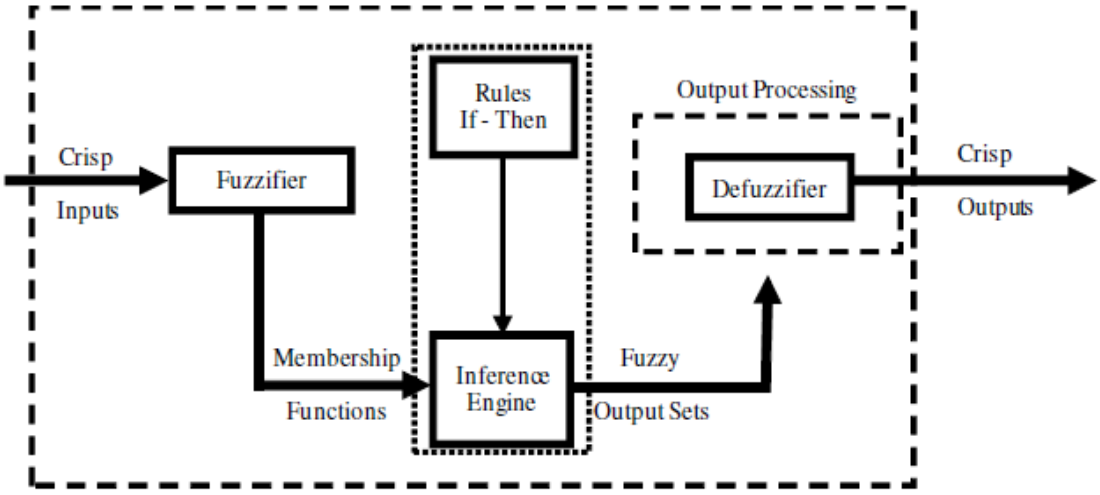


Figure 2.4: Fuzzy Logic System (FLS)

2.8 RELATED WORKS

Development of scheduling and control algorithms using intelligent systems has recently attracted attentions from researchers in different disciplines including operations researchers, control theorists and computer scientists. Researchers have employed various

fuzzy techniques to schedule tasks in the recent years in order to obtain an optimal performance, but this area of scheduling on multiprocessors is still an open problem because an ideal scheduling algorithm is yet to be discovered. The use of fuzzy logic approaches for scheduling in real-time environment is considered due to its ability to deal with uncertain and incomplete information and with multi-objective problems (Stallings, 2004). This review offers a brief, but informative, survey of a few reasonably successful attempts to extend popular scheduling algorithms to a fuzzy model.

The first paper reviewed, (Sabeghi *et al*., 2006), *Scheduling Non-Preemptive Periodic Tasks in Soft Real-Time Systems Using Fuzzy Inference*. The authors introduce a novel scheduling algorithm: highest fuzzy priority first (HFPF), whose performance is matched with a non-preemptive version of the Earliest Deadline First (EDF) algorithm. At any rate, the fuzzy inference engine takes in two inputs to be translated to fuzzy sets: *deadline* and *external priority*. External priority is the priority assigned to the task "from the outside world," assumed to be the user or task creator. The authors later point out that this priority is assigned according to a rate monotonic (RM) approach (the shorter the task, the higher its priority). In this approach, shorter jobs are more favoured than longer jobs which imply there is no fair sharing among the jobs.

A fuzzy scheduling approach to arrange real-time periodic and non-periodic tasks in multiprocessor systems (Mahdi *et al.*, 2007). The work focused on a real-time multiprocessor system with heterogeneous periodic and non-periodic tasks and compare performance and complexity using fuzzy scheduler algorithm. Task priority, deadline, required computation time and CPU time are considered as parameter for scheduling. The notion of laxity was employed to facilitate their computation (i.e. deadline minus computation_time). Trapezoid membership function was used because of its usability in

fuzzy dedicated hardware as they claimed (Ascia *et al*., 2001). The model consists of queue which offers tasks to dispatcher by their run-time priority order. Higher priority tasks have higher running probability. Their results show that the proposed fuzzy scheduler creates feasible schedules for homogeneous and heterogeneous tasks. According to their algorithm there is no clear difference between the computation time and CPU time, but they claim that they are two different entities and also their system do not considered overloading.

The model proposed in a *fuzzy-based CPU scheduling algorithm* (Shatha and Kasim, 2010), applied fuzzy logic in the design and implementation of a rule-based scheduling algorithm to solve the shortcoming of priority scheduling algorithms. Task priority and execution time was used as the input parameter for the fuzzy engine while shortest job first only considered the execution time. The results demonstrate that the average waiting time and the average turnaround time in the proposed model are better than that obtained using priority scheduling, and closed to that obtained from shortest-job-first (SJF) scheduling. Their work do not considered the scalability of system when the system is overloaded.

An improved fuzzy-based CPU scheduling algorithm for Real-Time Systems was developed which based on Mamdani-type architecture (Behera *et al*., 2012). The fuzzy scheduler architecture consists of burst time (BT), arrival time (AT) and priority (P) of the processes as the crisp input. Then membership of burst time (μb), priority(μp) and response ratio(μh) were computed. This model was compared with other existing algorithms such as SJF, Priority, and Highest Response Ratio Next (HRRN) and concluded that their model, minimizes the waiting time and turnaround time. However, their membership function formula is not clear to understand and their model is not a real

time system where tasks deadline have to be considered and the system load, but we reviewed this work due to the fact that fuzzy techniques was used as their system controller which may give better understanding in designing and implementation of our proposed fuzzy algorithm.

A fuzzy scheduler algorithm developed for real-time scheduling on multiprocessor (Sheo *et al.*, 2012), consists of two input parameters; external priority (Epriority) and deadline and one output (new priority) as depicted in figure 2.4. This scheduler used laxity interchangeably with deadline as a secondary parameter. The algorithms for the scheduler consists of nine fuzzy rules in the knowledge base. The algorithms were examined based on system load and other performance metric (such as deadline missed, response time and turnaround time) for system performance evaluation. In this approach, processors are placed in a cyclic version whereby highest priority tasks are assigned from the first processor to the last (say from P_1 to P_n) and continue in that order until all the tasks are assigned processor for their execution. There is no load balancing in this approach because some processors that finish execution on time will remain idle while others are heavily loaded, thus leads to low performance. However, their results show that using deadline as a fuzzy scheduling parameter in multiprocessor real-time scheduling is more promising than laxity because it gives better results than laxity.

A fuzzy Round Robin CPU scheduling algorithm was proposed to solve two major Round Robin' problems (i.e. choosing optimal time quantum (TQ) and context switching) (Bashir, 2013). The fuzzy inference system take in two inputs, number of tasks (N) and average burst time of all tasks (ABT) thereby produced an output which is the TQ. This TQ was used to schedule the tasks based on round robin policy to solve the first problem. Another fuzzy Inference system was later introduced with two inputs, Laxity and N, and

one output (Preemption Status) to solve the second problem. When the proposed model was compared with standard round robin, result shows that proposed approach outperformed the existing round robin.

However, having reviewed the related works, we discovered that the existing system in figure 2.5 considered few fuzzy rules which capture few external world of uncertainties. Also, the fuzzy scheduler has poor performance at higher system load in term of response time and number of tasks that meet their deadline after execution. As a result of this shortcoming, we extend this existing system by introducing new scheduling parameters and by developing new algorithm for tasks scheduling on multiprocessor. Chapter three will give more detail analysis of the new fuzzy scheduling algorithm design.

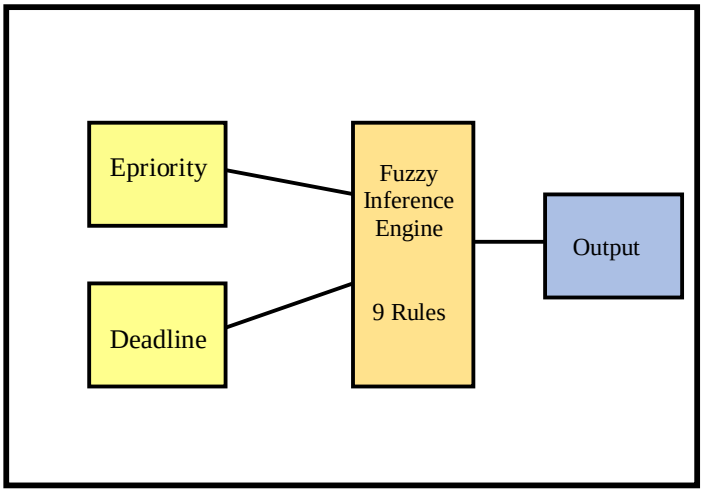


Figure 2.5: Existing Fuzzy Inference Model (Sheo *et al., 2012*)

CHAPTER THREE

NEW FUZZY SCHEDULING ALGORITHM (NFSA) DESIGN

3.0 INTRODUCTION

In order to efficiently schedule real-time tasks on multiprocessor and to be able to manage the processors efficiently, we need to consider the system performance criteria. The terms associated with each real-time task are:

i. **Processor Utilization:** It is the ratio of busy time of the processor to the total time passes for processes to finish. Processor needs to remain as busy as possible in order to perform the defined task (Robin, 2012).

ii. **Throughput:** It is defined as the number of processes that complete their execution per time unit.

iii. **Turnaround Time:** It is the time interval from submission of job till its completion. It includes actual processing time plus time spent on waiting for resources, including the processor.

iv. **Response Time:** It is the time taken from the time when a request was submitted till the first response is produced (Robin, 2012).

v. **Deadline Missed**: This is the measure of real-time tasks that do not meet their deadline after execution. Moreover, as we know that our work is based on soft real-time task, missing some deadline is still tolerable.

3.1 NEW FUZZY SCHEDULING ALGORITHM (NFSA) MODEL

CPU scheduling is the process of selecting a single task from a bunch of tasks from a ready queue for execution. This assignment is carried out by software known as a

scheduler. The existing fuzzy inference model (Sheo *et al.*, 2012), considered the external priority and deadline as the scheduling parameters as depicted earlier in figure 2.5 of the previous chapter. The selected parameters are not sufficient enough for fair and sufficient parameters for real-time tasks because the arrival time and the computation time (or CPU burst) of tasks were not considered. This computation time (or CPU burst) is used to measure processor performance in term of response time, turnaround time and utilization (i.e. how busy the processors are during execution).

Therefore our research work focused on the extension of the existing fuzzy model (Sheo *et al.*, 2012) by introducing two new fuzzy parameters and more fuzzy rules which captured possible external world uncertainties for fair and efficient scheduling on multiprocessor. The figure 3.1 depicts the new fuzzy inference model which consist of three major inputs as scheduling parameters, they are; *arrival time*, *computation time* and *deadline* and one output (Runtime priority) whose value determine the order of task execution. These parameters are generally attached with a real-time task when it is submitted to a fuzzy scheduling system for the purpose of execution.

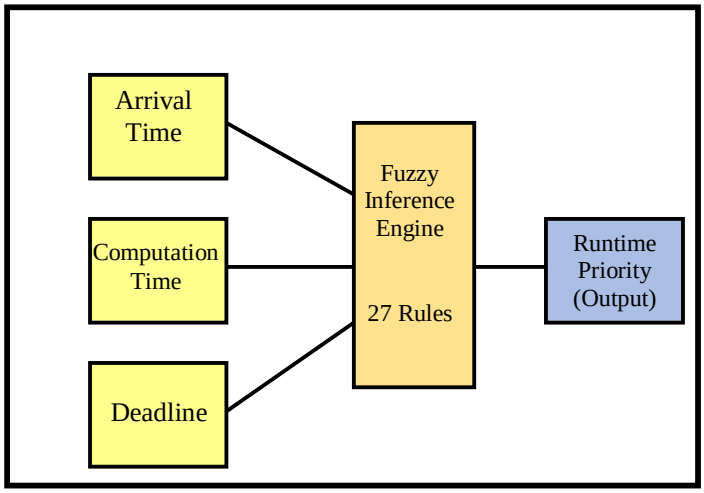


Figure 3.1: New Fuzzy Inference Model.

3.1.1 Defining the Inputs/Output Membership Function for NFSA

In this section we defined fuzzy input parameters with their linguistic categories (as described in section 2.5.3) such as *short, medium,* and *long* to be used for the arrival time, computation time and deadline of the real-time tasks. Linguistic category of each parameter is attached with every admitted task. The membership function distributions representing the linguistic concept for fuzzy inputs and output (runtime priority) were also defined. The fuzzy input parameters were fuzzified using triangular membership functions. Below are the linguistic categories and membership function distributions for the task' parameter defined;

i. **Arrival Time:** The table 3.1 shows the linguistic categories defined for the *arrival time* of all tasks. The defined Membership Function corresponding to *arrival time* of all tasks (ranges from 0-10) are depicted in figure 3.2.

Table 3.1: Fuzzy Variable Range for Arrival Time

Fuzzy Categories	**Crisp Input Range**
Early	0 – 5
Intermediate	0 – 10
Late	5 – 10

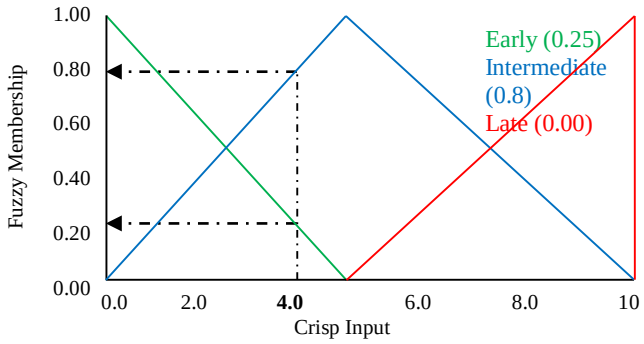


Figure 3.2: Membership Function corresponding to Arrival Time.

ii. **Computation Time:** Table 3.2 shows the linguistic categories defined for the *computation time* of all tasks. The defined Membership Function corresponding to *computation time* of all tasks (ranges from 1-20) are depicted in figure 3.3.

Table 3.2: Fuzzy Variable Range for Computation Time

Fuzzy Categories	**Crisp Input Range**
Short	1 – 10
Medium	1 – 20
Long	10 – 20

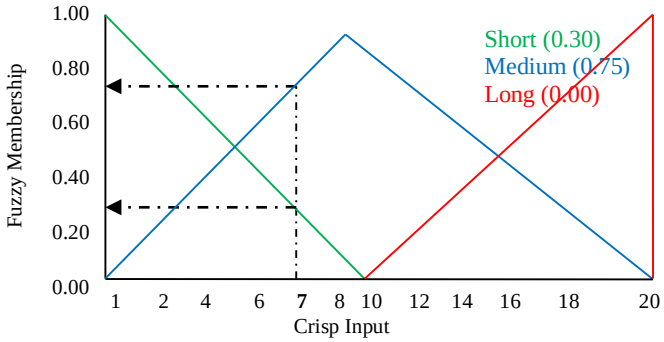


Figure 3.3: Membership Function corresponding to Computation Time.

iii. **Deadline:** Table 3.3 shows the linguistic categories defined for *deadline* of all tasks. The defined Membership Function for *deadline* of all tasks (ranges from 1-50) are depicted in figure 3.4.

Table 3.3: Fuzzy Variable Range for Deadline.

Fuzzy Categories	**Crisp Input Range**
Critical	1 – 25
Normal	1 – 50
Sufficient	25 – 50

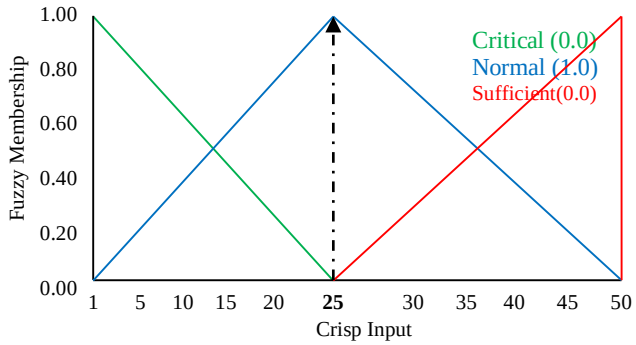


Figure 3.4: Membership Function corresponding to Deadline.

iv. **Runtime Priority *(Output)*:** Table 3.4 shows the linguistic categories defined for *runtime priority* of all tasks. The defined Membership Function for *runtime priority* of all tasks (ranges from 0-10) is depicted in figure 3.5.

Table 3.4: Fuzzy Variable Range for Runtime Priority.

Fuzzy Categories	**Crisp Input Range**
Low	0 – 5
Normal	0 – 10
High	5 – 10

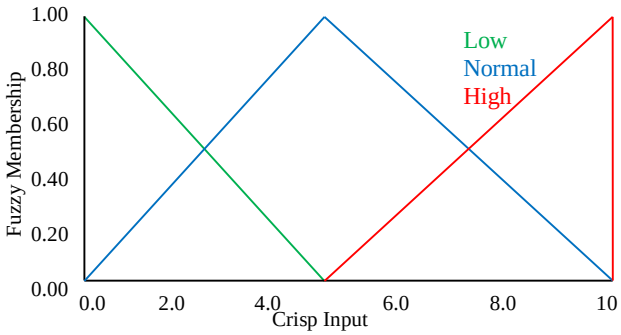


Figure 3.5: Membership Function corresponding to Runtime Priority.

3.1.2 Set-up Fuzzy Rule Base for NFSA

Now that the inputs and output membership functions have been defined in section 3.1.1, the next step is to set-up a fuzzy rule base on the basis of the number of linguistic categories defined for each of the input parameter in section 3.1.1. If *l, m, n* denote the number of linguistic categories defined for arrival time, computation time and deadline respectively, then the fuzzy rule base will have ($l \times m \times n$) rules. Therefore, this research work consists of twenty-seven 27 ($3 \times 3 \times 3$) rules of fuzzy logic. These rules usually take the form of IF – THEN rules and are obtained from human expert.

The table 3.5 shows five (5) of this rules. The complete rules were presented in appendix I. As shown in table 3.5, the first rule reads thus;

IF *arrival time* is early and *computation time* is short and *deadline* is critical,

then *runtime priority* is high.

Table 3.5: Fuzzy Rules for NFSA

S.No.	Arrival Time	Computation Time	Deadline	Runtime Priority
i.	Early	Short	Critical	High
ii.	Intermediate	Short	Sufficient	Normal
iii.	Late	Short	Critical	High
iv.	Early	Medium	Critical	High
v.	Late	Long	Sufficient	Low

3.1.3 Defuzzified the Output for NFSA

However, since the fuzzy rules have been stated, the next step is to use a standard fuzzy union to aggregate the membership functions of the three parameters attached with each task based on the fuzzy rules and defuzzified the output using centroid method. This gives

a crisp output value for each task. This crisp output value is the runtime priority for task execution.

Let assume a task has the following parameters; arrival time = 4ms, computation time = 7ms and its deadline = 25ms. These are the crisp inputs entered from the outside world. As depicted in figure 3.2 to 3.4, the dotted arrows indicate the various membership value of each of the crisp inputs. When we fuzzified these inputs, the following membership values were obtained:

i. When crisp input *Arrival time* = 4.0,

Its fuzzy outputs with membership values are;

Early: 0.25

Intermediate: 0.80

Late: 0.00

ii. When crisp input *Computation time* =7.0,

Its fuzzy outputs with membership values are;

Short: 0.30

Medium: 0.75

Long: 0.00

iii. When crisp input *Deadline* =25.0,

Its fuzzy outputs with membership values are;

Critical: 0.00

Normal: 1.00

Sufficient: 0.00

Out of the twenty-seven (27) rules generated, the exemplified results triggered only four (4) of these rules;

i. If *Arrival time* is Early (0.25) AND *Computation time* is Short (0.30) AND *Deadline* is Normal (1.00), THEN *Runtime priority* is High.

ii. If *Arrival time* is Early (0.25) AND *Computation time* is Medium (0.75) AND *Deadline* is Normal (1.00), THEN *Runtime priority* is Normal.

iii. If *Arrival time* is Intermediate (0.8) AND *Computation time* is Short (0.30) AND *Deadline* is Normal (1.00), THEN *Runtime priority* is High.

iv. If *Arrival time* is Intermediate (0.80) AND *Computation time* is Medium (0.75) AND *Deadline* is Normal (1.00), THEN *Runtime priority* is Normal.

In order to determine the membership value of each output generated by the four (4) rules, the minimum AND (or conjunction) operator is used. This minimum operator simply takes the minimum membership value when compared with each other. Therefore, for each of the rules, the following firing strengths were generated:

i. $(0.25) \cap (0.30) \cap (1.00) = 0.25$

ii. $(0.25) \cap (0.75) \cap (1.00) = 0.25$

iii. $(0.80) \cap (0.30) \cap (1.00) = 0.30$

iv. $(0.80) \cap (0.75) \cap (1.00) = 0.75$

By using the fuzzy rule base and the firing strengths generated previously, the fuzzy inference engine finally generate the following output values:

i. High (0.25)

ii. Normal (0.25)

iii. High (0.30)

iv. Normal (0.75)

These output values are used and a particular triangular membership function to the strength level is filled as depicted in figure 3.6 to 3.9. For example, in the first rule, the

triangular membership function, High to the level 0.25 is filled and it top (above 0.25) is cut off (truncated). The same procedure is applied to the other rules. Finally, the truncated figures are combined together based on overlapping composition method as shown in figure 3.10 to generate crisp output value for the centre of gravity (i.e. runtime priority).

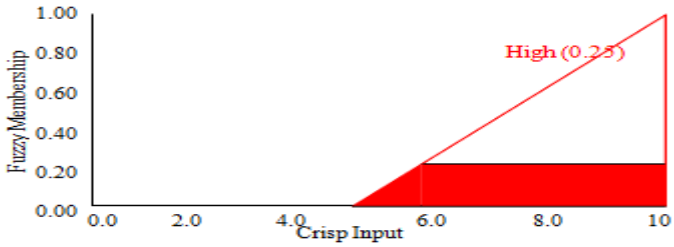


Figure 3.6: Triangular Membership Function for High at 0.25

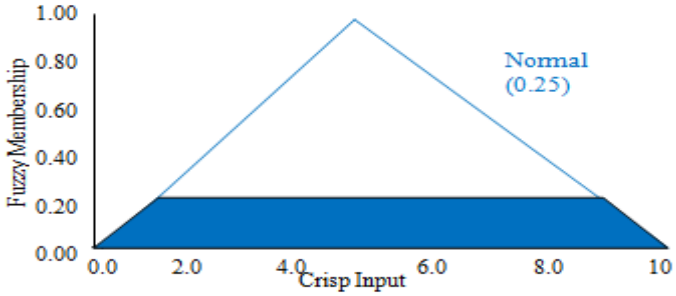


Figure 3.7: Triangular Membership Function for Normal at 0.25

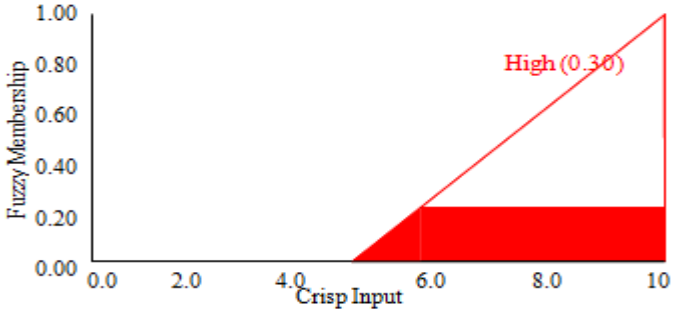


Figure 3.8: Triangular Membership Function for High at 0.30

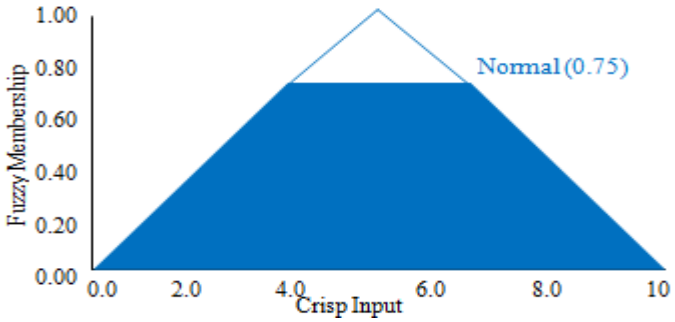


Figure 3.9: Triangular Membership Function for Normal at 0.75

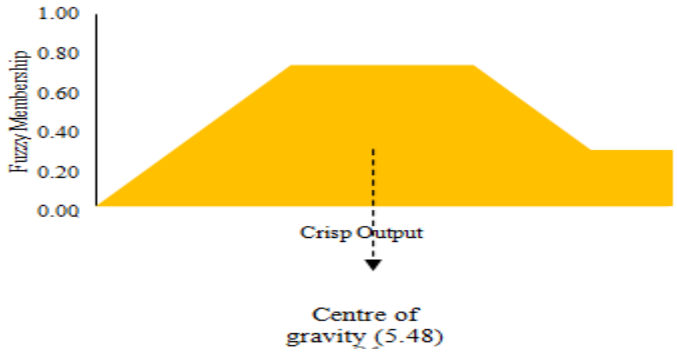


Figure 3.10: Defuzzification with Centroid Approach

The centroid approach have defuzzified the various membership values and finally generates the crisp output which is 5.48 as centre of gravity. This value is the crisp *runtime priority* for which tasks are scheduled (ordered) into the *runtime priority* (ready) queue, for execution on processors.

3.2 CASE ASSUMPTION

In a real-time processing environment, tasks of different characteristics are submitted to the multiprocessor by the fuzzy scheduler as described in the previous chapter, this research assumed a system comprising of 5 to 25000 real-time tasks to be assigned to multiprocessor based on the EFA and NFSA algorithm. In order to facilitate the feasibility

analysis of this work, several assumptions are used which are described in the following paragraph:

Let T_i represents a periodic task, and U represents a set of periodic tasks;

i. A task cannot suspend itself (i.e. no pre-emption).
ii. All tasks T_i are independent (i.e. there is no relation between the tasks from the same set U).
iii. All tasks T_i have deadline D_i equal to their next request time (period), (Liu and Layland, 1973).
iv. All the processors are identical.
v. All tasks T_i evacuate the arrival queue at the same time into fuzzy inference engine (Sheo *et al.,* 2012).
vi. All tasks T_i are activated in the *runtime priority* (ready) *queue.* Therefore, all tasks in *runtime priority queue* arrive at the processors node at time t = 0.0ms (Sheo *et al.*, 2012).

3.3 NFSA ARCHITECTURE

As described earlier in section 3.1, the new fuzzy inference model is the brain box in our fuzzy scheduler which produces *runtime priority* as the output for task execution on multiprocessor. The figure 3.11 depicts the architectural model of the new fuzzy scheduler which consists of four processing phases $\lambda_{i(i=1,2,3,4)}$ for task execution; *arrival queue*, the new *fuzzy inference engine* (as the system controller), *runtime priority queue* and the *multiprocessor*. A task pool *N* was initialized in the arrival queue as a set containing various periodic tasks *T* with distinct *arrival time a*, *computation time c* and *deadline d*.

$N = \{T_1, T_2, T_3, \ldots, T_n\}$ (3.1)

Arrival-time a: $a_1, a_2, a_3, \ldots, a_n$ for every a_i associated with T_i at λ_0. (3.2)

Computation-time c: $c_1, c_2, c_3, \ldots, c_n$ for every c_i associated with T_i at λ_0. (3.3)

Deadline d: $d_1, d_2, d_3 \ldots, d_n$ for every d_i associated with T_i at λ_0. (3.4)

i. **Arrival Queue:** As the new tasks arrived at λ_0, they are first of all stored in an arrival queue with their parameters (arrival time, computation time and deadline). After a fixed period of time these tasks are fed into the fuzzy inference engine.

ii. **Fuzzy Inference Engine:** The inference engine takes in the tasks parameters at λ_1 and undergoes all the procedure described in section 3.1.1 to 3.1.3 and output a crisp value as the *runtime priority*.

iii. **Runtime Priority Queue:** At λ_2 the fuzzy scheduler ordered the tasks into runtime priority (ready) queue based on the crisp output values generated by the fuzzy inference engine in descending order.

iv. **Multiprocessor:** Finally, the scheduler assigns the first set of tasks from the highest to the lowest *runtime priority* to the number of available processors and execute for the first cycle of assignments at λ_3. The scheduler searches for any available processor (i.e. processor with least accumulated computation time) and assigns the next task to it for execution after the execution of the first set of tasks assigned initially. This procedure are repeated until all the tasks are been assigned processor for their execution, then the system state is updated.

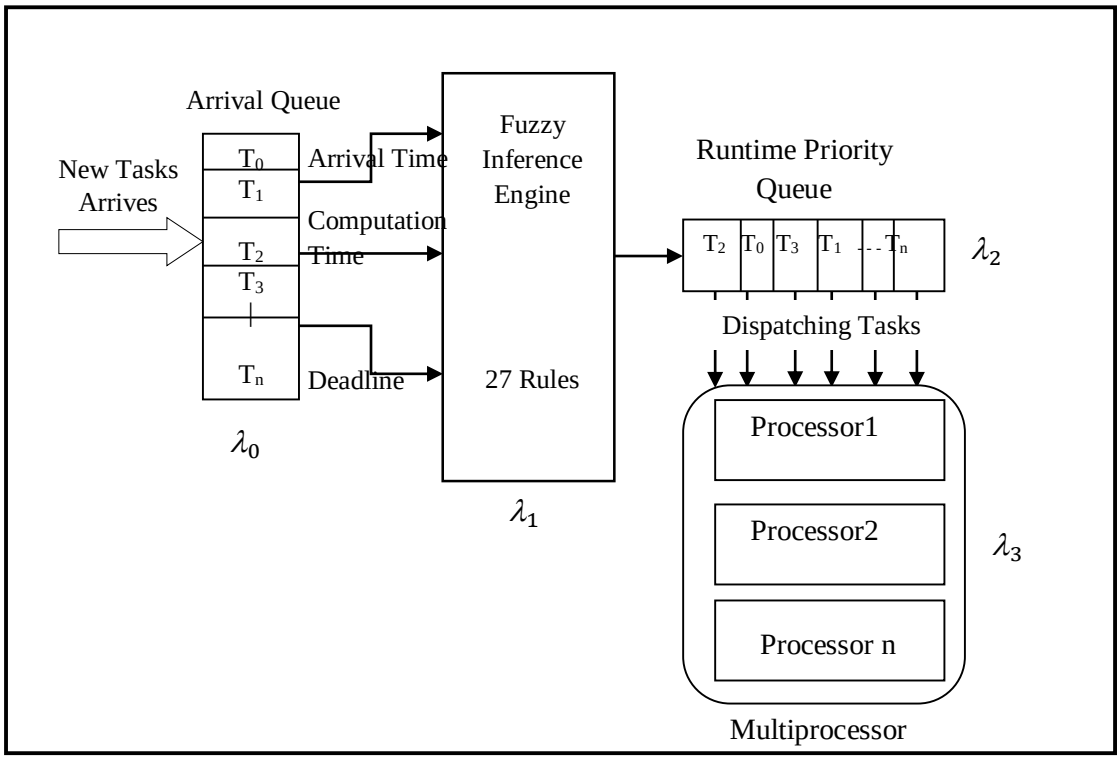


Figure 3.11: New Fuzzy Scheduler Architecture.

3.4 NFSA ALGORITHM

For the Fuzzy Scheduler do the followings:

Loop

i. Initialize a task pool *N* in an arrival queue with task parameter $T_i(a_i, c_i, d_i)$ at λ_0. Where (*i=1,2,3,.....,n*).

ii. For each ready task, feed in task parameter $T_i(a_i, c_i, d_i)$ into fuzzy inference engine at λ_1. Considered the fuzzy inference engine output as the *runtime priority* r_i for each task execution.

iii. Sort all tasks T_i in descending order of *runtime priority* r_i into the priority (ready) queue at λ_2.

iv. Since all processors are idle at the initial stage,

3.1 Assign processor $P_{i(i:1,2,3,\ldots m)}$to the first set of tasks $T_{i(i:1,2,3,\ldots,m)}$ with highest *runtime priority* r_i and execute at λ_3. For $(n \geq m)$.

3.2 Search through all the processor $P_{i(i:1,2,3,\ldots m)}$with the least computation weight (cw).

If $cw(P_i) < cw\left(P_{(i+1)}\right)$,

Assign P_i to $T_{(m+1)}$ and execute

Else,

Assign $P_{(i+1)}$ to $T_{(m+1)}$ and execute.

v. Update the system states.

End Loop.

3.5 FLOW CHART FOR NFSA

This section depict the flow chart for NFSA as described in section 3.2

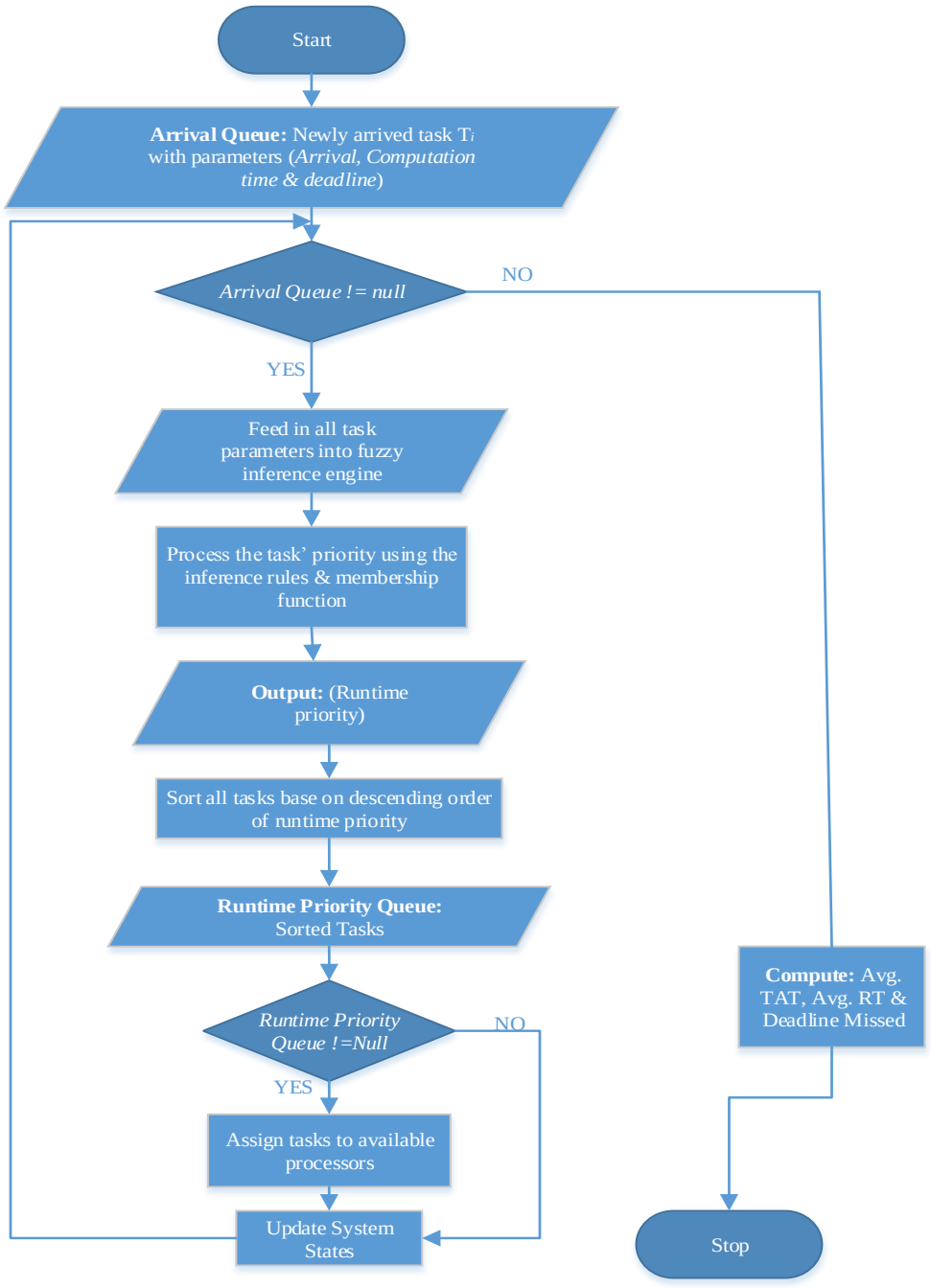


Figure 3.12: Flow Chart for NFSA

CHAPTER FOUR

IMPLEMENTATION AND TEST OF THE NFSA

4.0 INTRODUCTION

In the implementation of NFSA model, different cases were tested after the model was designed in java to create a graphical user interface where number of tasks and number of processors were entered and the ranges for computation time are specified. The collective results for each running of the program is written to an external text file called out.txt where sample results of proposed algorithm are analysed and compared with the results obtained from the existing algorithms.

As earlier described in chapter two, Load factor is considered and it is the summation of the ratio of all task computation time to deadline (equation 2.1). In real-time systems, load factor is used as a measure of processor utilization. Therefore, this research work uses load factor to determine when the system is under normal load and when is overloaded. However, since we are dealing with real-time systems scheduling parameters, *arrival time* of each task was generated using Poisson distribution, while *computation time* and *deadline* of each task were generated using normal and uniform distributions.

4.1 ILLUSTRATIVE EXAMPLE

In this section, a simple illustrative example was presented in which five (5) real-time tasks were generated using uniform distribution with the computation time ranges from 2ms to 20ms. However, following all steps of proposed NFSA described in chapter three, the performance metrics of NFSA and EFA will be evaluated in terms of *average response time (ART), average turnaround time (ATAT)* and *number of deadline missed*

(DLM) and compared the performances of both systems based on the result that will be generated .

As shown in table 4.1, three (3) processors and five (5) tasks were randomly generated with arrival time range 0 - 10ms, computation time range 1 - 10ms, external priority range 1 – 12 and deadline range 5 – 30ms. In this section, system under normal load was considered for the purpose of illustrative example. Other cases like when the system is overloaded will be examine and analysed graphically later in this chapter. For simplicity arrival time are defined as *AT*, computation time as *CT* and external priority as *Epriority*.

Table 4.1: Tasks Generated under Normal Load for Three (3) Processors.

Tasks ID	**AT**	**CT**	**Deadline**	**Epriority**
T_0	0.0	7.0	14.0	2.0
T_1	0.0	3.0	7.0	2.0
T_2	5.0	5.0	17.0	7.0
T_3	7.0	4.0	19.0	4.0
T_4	8.0	8.0	15.0	7.0

At a predefined period of time (say 30ms) a set of tasks input parameters (AT, CT, Deadline and Epriority) from arrival queue are fed into the NFSA system using EFA and NFSA scheduling algorithms. Recall that EFA described in chapter two only uses *Epriority* and *Deadline* as their scheduling input parameters while NFSA uses *AT*, *CT* and *Deadline* as scheduling input parameters. As described in chapter three, table 4.2 shows the output results generated by both algorithms. The EFA and NFSA columns were the fuzzy output (runtime priority) generated from the fuzzy inference engine which determines the order of processor assignment to each task for execution.

Table 4.2: Output (Runtime Priority) of both EFA and NFSA

Tasks ID	**AT**	**CT**	**Deadline**	**Epriority**	**EFA** (Output)	**NFSA** (Output)
T_0	0.0	7.0	14.0	2.0	3.57	5.48
T_1	0.0	3.0	7.0	2.0	3.85	6.37
T_2	5.0	5.0	17.0	7.0	7.47	5.75
T_3	7.0	4.0	19.0	4.0	4.30	5.41
T_4	8.0	8.0	15.0	7.0	7.50	4.11

The table 4.3a and 4.3b show how fuzzy scheduler of both algorithms sorted the entire tasks based on the value of runtime priority from the highest to the lowest value for execution on multiprocessor.

Table 4.3a: EFA Sorted Tasks

Tasks ID	**EFA** (Output)
T_4	7.50
T_2	7.47
T_3	4.30
T_1	3.85
T_0	3.57

Table 4.3b: NFSA Sorted Tasks.

Tasks ID	**NFSA** (Output)
T_1	6.37
T_2	5.75
T_0	5.48
T_3	5.41
T_4	4.11

4.1.1 Performance Evaluation Metric of EFA

In this section, tasks were scheduled on multiprocessor based on the sorted order of each algorithm as depicted in table 4.3a and 4.3b. The figure 4.1 depicts Gantt chart of EFA mapping each of the 5 tasks to the 3 available processors. According to the existing model

(Sheo *et al.*, 2012), EFA only take in two scheduling parameters (Epriority and Deadline) for it execution. Cyclic assignments were employed for multiprocessor and tasks mapping. Their processor assignment is cyclic in the sense that their fuzzy scheduler assign tasks from the first processor to the last and repeat the same assignment process until all tasks are been assigned to processor. This approach may lead to load imbalance where some processors are idle while others maybe overloaded as a result of cyclic assignment. However, this may result to low response time, turnaround time and may also lead to higher number of tasks miss their deadline.

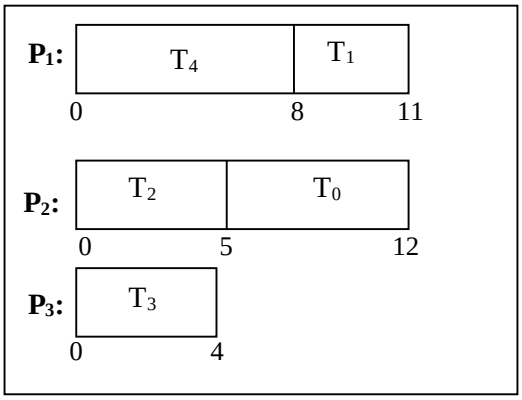


Figure 4.1: Gantt Chart EFA Tasks Assignment on Three (3) Processors

As stated earlier that average response time (*ART*), average turnaround time (*ATAT*) and number of tasks missed deadline (*DLM*) will be evaluated. Therefore, based on the assumptions made earlier in section 3.2, the evaluation of this performance metric will follow the assumption that all tasks are only activated after they are moved in to the *runtime priority (ready) queue* for scheduling. Based on this assumption, all tasks arrived *runtime priority queue* at time t = 0.0ms.

EFA: *For each process, Response Time (RT) = Time first Scheduled (in ready queue) – Arrival time*

Average Response Time (ART) = Sum of all Tasks Response Time/ Total numbers of tasks.

Recall that Arrival Time of all task in a runtime priority queue at time t = 0.0ms therefore, from figure 4.1;

T_4: (0-0) = 0, where T_4 is the first task that arrived at time t = 0 on the first processor P_1 thus didn't wait before execution begin.

T_2: (0-0) = 0, where T_2 arrived at time t = 0 and was scheduled on the second processor P_2 thus didn't wait before execution begin.

T_3: (0-0) = 0, where T_3 arrived at time t = 0 and was scheduled on the third processor P_3 thus didn't wait before execution begin.

$T_{1:}$ (8-0) = 8, since EFA is using cyclic assignment T_1 was scheduled on processor P_1 and waited for T_4 to finish execution at t = 8.0ms likewise,

$T_{0:}$ (5-0) = 5, where T_0 was scheduled on processor P_2 and waited for T_2 to finish execution at t = 5.0ms

Average Response Time (ART) = Sum of all Tasks Response Time/ Total numbers of tasks.

Thus ***ART*** = (0 + 0 + 0 + 8 + 5) / 5

= 13/ 5

= 2.6ms

Turn Around Time (TAT) = Overall Time a task spent in the system (after activation)

= Time of task completion – Arrival Time (at time t = 0)

Average Turn-Around Time = Sum of each task / Total numbers of Tasks.

Thus, TAT in the system will be:

T_4: (8-0) = 8, T_2: (5-0) = 5, T_3: (4-0) = 4, $T_{1:}$ (11-0) = 11, $T_{0:}$ (12-0) = 12

ATAT *= (8 + 5 + 4 + 11 + 12) / 5*

= 40/5

= 8ms

Finally, number of tasks that missed deadline was also evaluated thus;

$\forall\, T_{i(i=0,1,2,\ldots,n)}; DLM = 1, iff\ Deadline < Turnaround\ Time$

This simply implies that *DLM* return a Boolean value *True* if each task finished its execution after the maximum specified time (Deadline), otherwise *False*. The table 4.4 shows that only one task T_1 missed its deadline because it finished execution 4.0ms after its deadline.

Table 4.4: EFA Deadline Missed

Tasks ID	**Deadline**	**TAT**	**DLM (Deadline<TAT?)**
T_4	15.0	8.0	F
T_2	17.0	5.0	F
T_3	19.0	4.0	F
T_1	7.0	11	T
T_0	14.0	12	F

4.1.2 Performance Evaluation Metric of NFSA

In NFSA approach, tasks are scheduled on processors based on the sorted order of NFSA algorithm as depicted in table 4.3b. The same assumptions stated in section 3.2 still valid for NFSA except in the assignment of processors to tasks. Unlike EFA, NFSA assigned processors to task based on the algorithm described in chapter three. The same example used for EFA in the previous section was also used in NFSA in order to compare the performance of both algorithms.

As depicted in figure 4.2, the Gantt chart of NFSA mapping each of the 5 tasks to the 3 available processors. The NFSA scheduler first of all scheduled equal number of task to

processors at time t = 0.0ms. After each processor have been assigned to a task in the first assignment, the NFSA scheduler searches for processor with the minimum execution weight (or shortest execution time) and assigned it to the next task in *runtime priority (ready) queue* for the next execution. This procedure continues until all the tasks have scheduled on multiprocessor. This approach successfully balances tasks on the processors. NFSA scheduler also keep track of tasks which arrived late since arrival time is involved in NFSA scheduling algorithm in order to avoid starvation by fairly scheduling them for execution based on the fuzzy rules stated in chapter three.

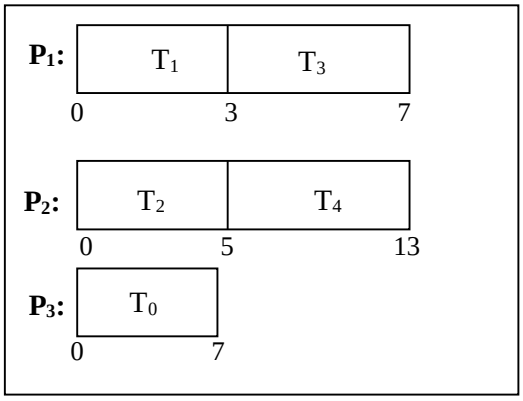


Figure 4.2: Gantt Chart NFSA Tasks Assignment on Three (3) Processors

NFSA: *For each process,*

Response Time (RT) = Time first Scheduled (in ready queue) – Arrival time

Recall that Arrival Time of all task in a runtime priority queue at time t = 0.0ms therefore, from figure 4.2;

T_1: (0-0) = 0, where T_1 is the first task that arrived at time t = 0 on the first processor P_1 thus didn't wait before execution begin.

T_2: (0-0) = 0, where T_2 arrived at time t = 0 and was scheduled on the second processor P_2 thus didn't wait before execution begin.

T_0: (0-0) = 0, where T_0 arrived at time t = 0 and was scheduled on the third processor P_3 thus didn't wait before execution begin.

$T_{3:}$ (3-0) = 3, recall that NFSA searches for processor with least execution time after the first cycle of the assignment. Therefore, processor P_1 has the least execution time 3ms in the first cycle of assignment thus, T_3 was scheduled on processor P_1 and waited for T_1 to finish execution at t = 3.0ms likewise,

$T_{4:}$ (5-0) = 5, where T_4 was scheduled on processor P_2 been the processor with least execution time of 5ms in the second cycle of assignment and waited for T_2 to finish execution at t = 5.0ms

Therefore, *Average Response Time (ART) = Sum of all Tasks Response Time/ Total numbers of tasks.*

Thus ***ART*** = (0 + 0 + 0 + 3 + 5) / 5

= 8/ 5

= 1.6ms

Turn Around Time (TAT) = Overall Time a task spent in the system (after activation)

= Time of task completion – Arrival Time (at time t = 0)

Average Turn-Around Time = Sum of each task / Total numbers of Tasks.

Thus, TAT in the system will be:

T_1: (3-0) = 3, T_2: (5-0) = 5, T_0: (7-0) = 7, $T_{3:}$ (7-0) = 7, $T_{4:}$ (13-0) = 13

ATAT *= (3 + 5 + 7 + 7 + 13) / 5*

= 35/5

= 7.0ms

Finally, number of tasks that missed deadline was also evaluated. The table 4.5 shows that no task misses its deadline in NFSA.

Table 4.5: NFSA Deadline Missed

Tasks ID	Deadline	TAT	DLM (Deadline<TAT?)
T_1	7.0	3.0	F
T_2	17.0	5.0	F
T_0	14	7	F
T_3	19	7	F
T_4	15	13	F

4.1.3 Comparing NFSA with EFA

Based on the results of illustrated example in the previous sections, both algorithms performance were compared with each other. The results in table 4.6 proved that NFSA has better performance by minimised the ATAT, ART and number of task that missed deadline. These was achieved as a result of increment in fuzzy rules which captured possible scenario in the system model and the way NFSA algorithm scheduled tasks on multiprocessors by trying to balance the processor's load on every assignment cycle.

Table 4.6: Comparing EFA and NFSA

Algorithms	ATAT	ART	Deadline Missed
EFA	8.0	2.6	1
NFSA	7.0	1.6	0

4.2 SYSTEM REQUIREMENT

In this research work, listed below are the system requirements for the implementation of the NFSA model.

a. Hardware Requirements

b. Software Requirements

a. **Hardware Requirements:** The following hardware are required for the implementation of NFSA model;

   i. Computer/Processor: AMD E-300 APU, Dual core, 1 gigahertz (GHz) of 32-bit (x86) or 64-bit (x64) processor.

   ii. Memory (RAM): 1 gigabyte (GB) RAM (32-bit) or 2 GB RAM (64-bit).

   iii. Hard Disk: 1 GB free disk space (32-bit) or (64-bit).

   iv. Graphics hardware: Microsoft Directx 9 (API) 128 Megabyte (MB) graphics memory.

   v. Operating System: Window 7 Home Premium.

b. **Software Requirements:** Listed below are the software requirement for NFSA model;

   i. JDK 6 Update 13 or later version

   ii. NetBeans IDE 6.9.1

4.3 USER INTERFACE

The graphical user interface (GUI) is very simple and easy to use by any users. As depicted in figure 4.3, the interface consists of labels, text fields, buttons and dialog box which are programmed and designed using java programming language. The *generate parameters* button generates required number of tasks parameters as entered in the text fields to an external file called *data.txt.* After all the parameters were generated successfully into an external file by the click of *generate parameters* button, a dialogue box pops up displays *task scheduling parameters were generated successfully* as depicted in figure 4.3.

When *simulate* button is click, the main program triggered EFA and NFSA algorithm to schedule tasks on multiprocessor and stored the results on external files called *olddata.txt* and *newdata.txt* respectively. Finally, the *show results* button displays the result analysis of both algorithms from the external files into a graphical panel as depicted in figure 4.4. Elaborate details of the source code are presented in the appendix III.

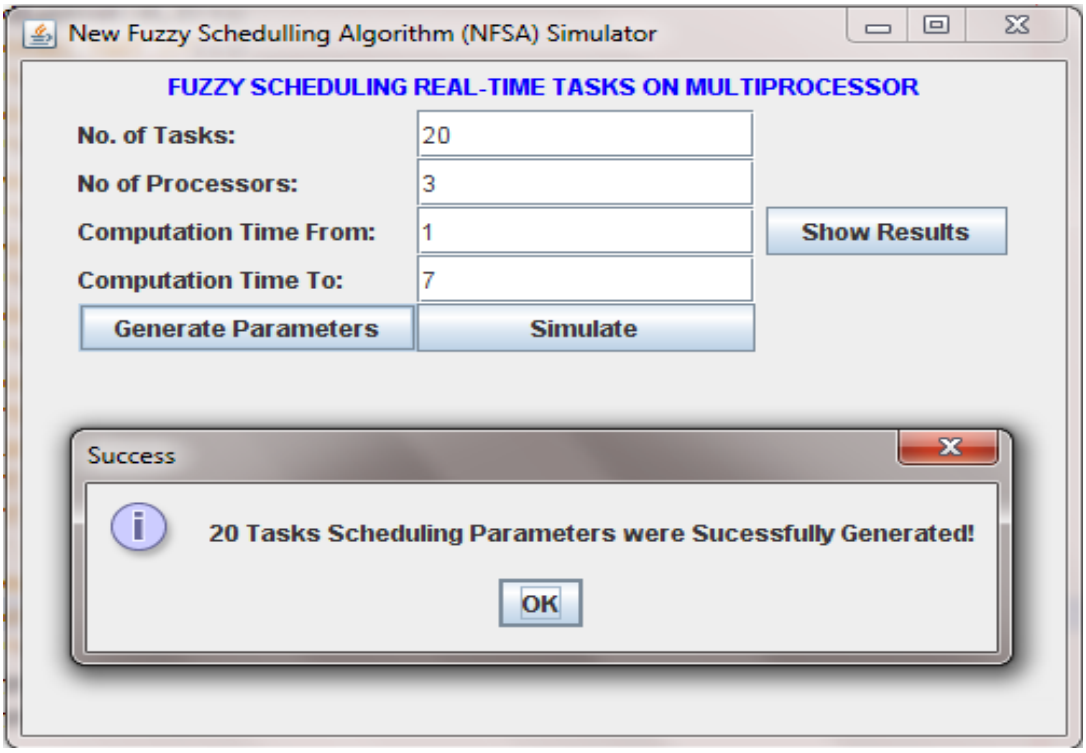


Figure 4.3: NFSA Users Interface.

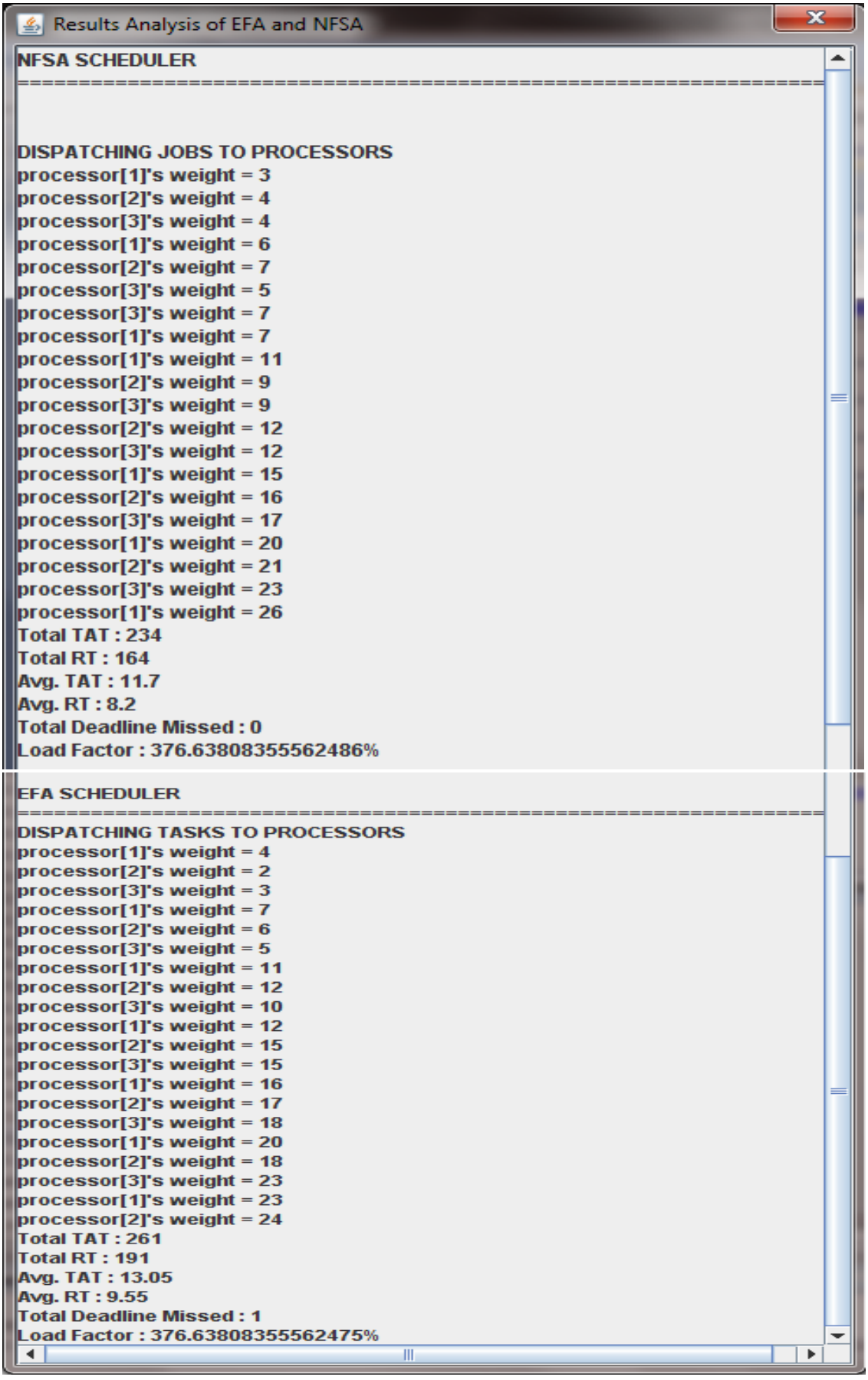


Figure 4.4: Results Analysis of EFA and NFSA after Simulation.

4.4 RESULTS DISCUSSION AND CONCLUSION

The functionality of operating system which serves as an interface between the user and hardware components depends on its task/process scheduling when it has minimal response time, minimal average turnaround time and reduces deadline miss (Galvin, 2004). To achieve all the listed advantages, this research compared the EFA results with the proposed NFSA results for performance evaluation. The performance metrics used were carefully chosen in order to reflect the real characteristics of a real-time system. As stated in the previous sections, the performance metric considered are; *average turnaround time (ATAT), average response time (ART)* and *number of deadline missed* which is an influential metric in scheduling algorithms for soft real-time systems.

However, the numbers of processors considered in this research are 3, 10 and 100 as the multiprocessor for simulation. Computation time ranges 1 – 25ms were applied across the processors and 5 – 25000 tasks were randomly generated with different load factor. The task parameters *arrival time* was generated using Poisson distribution while *computation time* and *deadline* were generated using uniform and normal distribution. In this research several test cases were simulated and the behaviours of both algorithms were compared with each other to determine the strength of the proposed algorithm.

4.4.1 Graphical Representation of Average Turnaround Time.

The graphical results of EFA and proposed NFSA were depicted in the following figures. The figure 4.5 depicts the average turnaround time of both algorithms using three processors with number of tasks generated ranges from 5 to 100 tasks. The graph shows that both algorithms has similar performance when the system load is minimal 0.32 to 0.39 but as the system load increases, the proposed NFSA have better performance even when the system is heavily loaded at 1.08 to 3.21.

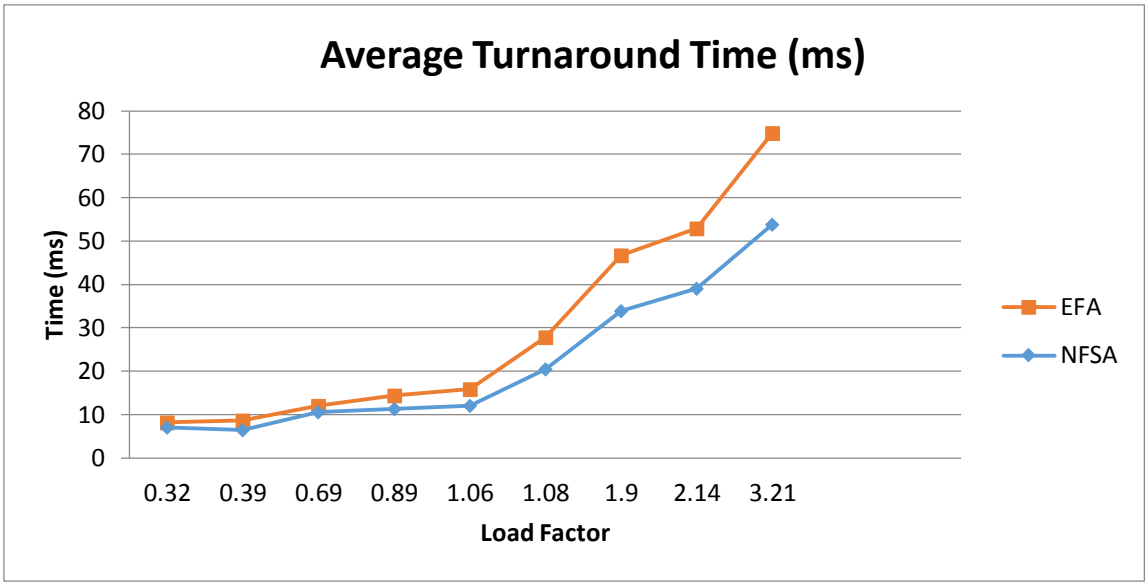


Figure 4.5: Average Turnaround Time on 3 Processors.

As depicted in figure 4.6, the graph showing the average turnaround time for 10 processors with number of tasks ranges from 12 to 500. The computation time ranges from 1 to 15ms. The graph shows that when system load is very low, both algorithms similar performance at 0.12 to 0.64 but as the load increases the proposed NFSA shows better performance.

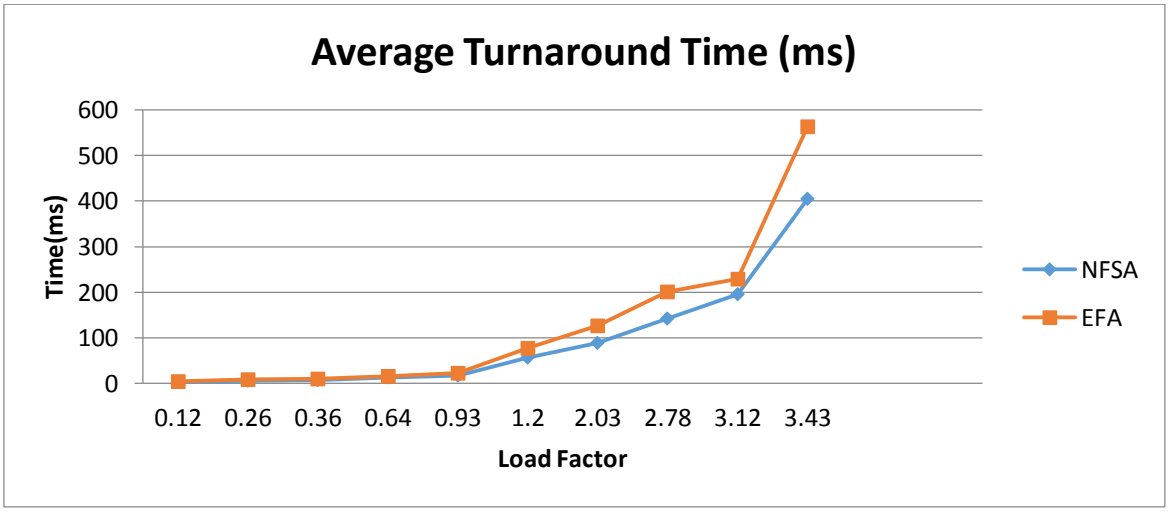


Figure 4.6: Average Turnaround Time on 10 Processors

From figure 4.7, the number of processors was increased to 100 and we carefully considered the system when is heavily loaded with tasks generated ranges from 200 to 20000 tasks with computation time ranges from 1 to 25ms. The proposed NFSA is scalable even as the system load increases from 0.45 to 3.48.

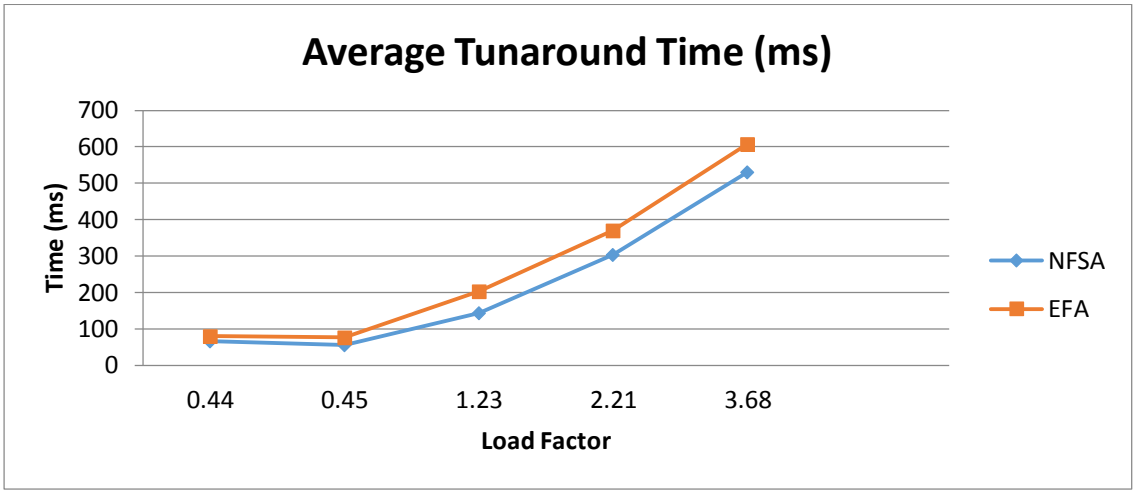


Figure 4.7: Average Turnaround Time on 100 Processors.

4.4.2 Graphical Representation of Average Response Time.

This section gives the graphical representation of average response time results obtained from both algorithms. The data used for this graph are presented in appendix II. The same number of processors was considered as used in the previous section. The figure 4.8 depicts the average response time for 3 processors. The generated tasks range from 5 to 100 tasks with computation time ranges from 1 to 10ms. The result shows that the proposed NFSA is tardier than EFA algorithm thus NFSA have better response time.

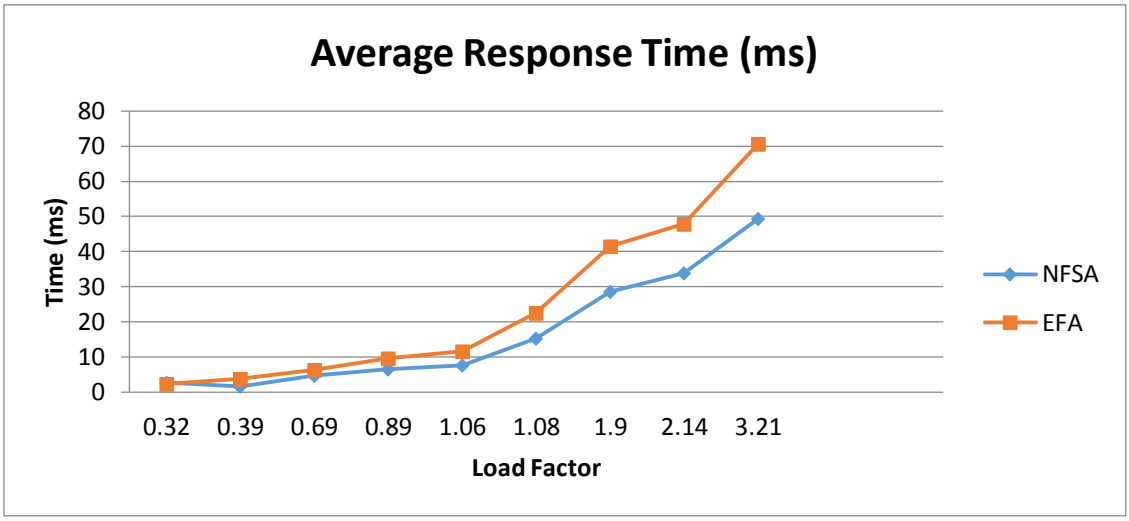


Figure 4.8: Average Response Time on 3 Processors

From figure 4.9, the system was simulated for 10 processors with the same number of tasks and computation time used in section 4.5.1. It was observed that the response time differs when the system load is about 1.0, this proved that the proposed algorithm NFSA have better performance at higher system load.

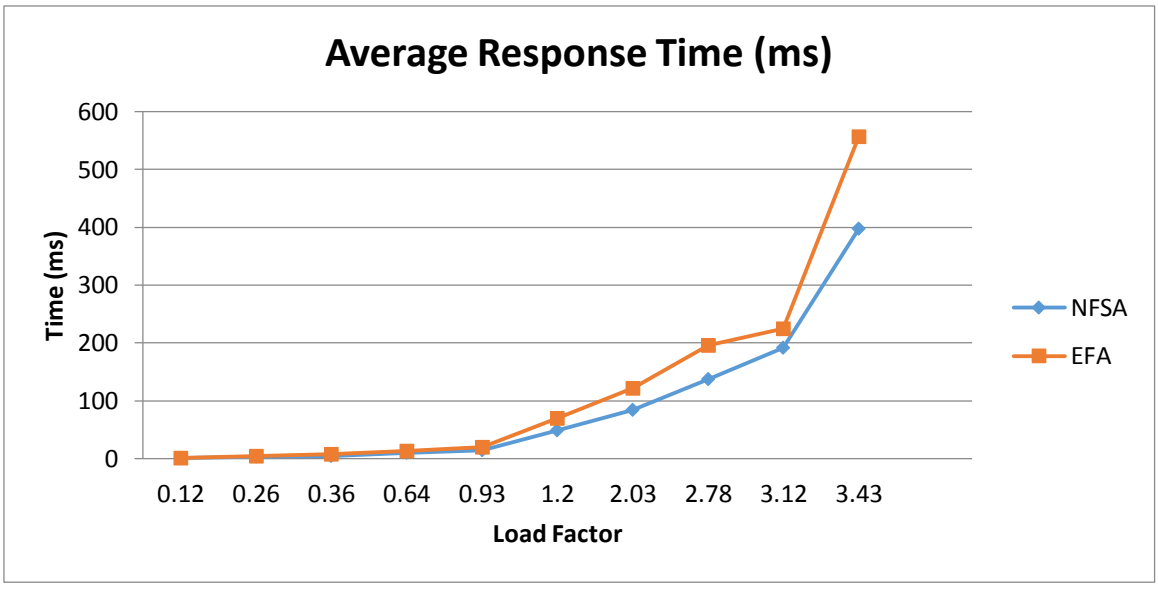


Figure 4.9: Average Response Time on 10 Processors

Finally, the number of processors was increased to 100 in order to affirm the claim that the proposed algorithm has better performance at higher load. Therefore, figure 4.10

depicts the average response time of 100 processors using both algorithms. The system is loaded with higher number of generated task from 1000 to 25000 with computation time ranges from 2 to 25ms. At initial point (0.44), it was discovered that both algorithms have similar performance since the system is still under normal loads, but as the system is heavily loaded at 0.45 to 3.68 the proposed NFSA proved to have better performance than EFA. However, this results shows that the proposed NFSA always have better response time at higher system load.

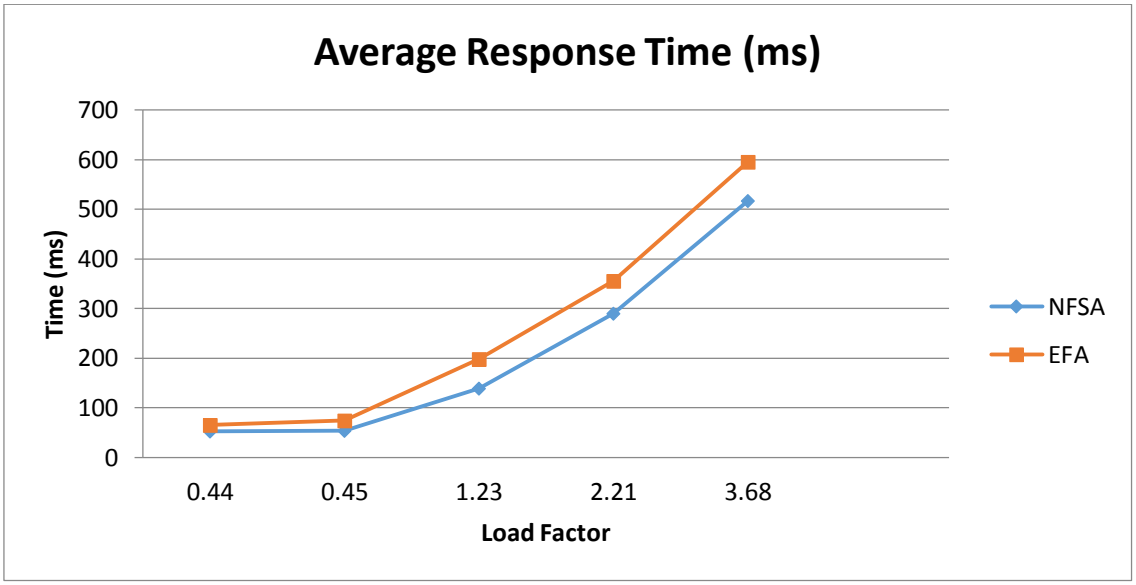


Figure 4.10: Average Response Time on 100 Processors.

4.4.3 Graphical Representation of Number of Tasks Miss Deadline

As stated earlier that 3, 10 and 100 processors were considered as multiprocessor for the simulation and 1034 test cases were carried out in which better results were selected for evaluation.

The results depicted from figure 4.11 to figure 4.13 proved that the proposed NFSA is standard as compared to an existing algorithm EFA. The proposed NFSA is standard with EFA in the sense that when a system is under normal loads and does not exceeds its utilization bound as discussed in chapter two; its tasks are schedulable and will also meet

their deadline (Liu and Layland, 1973). Therefore, it was observed that under normal load both algorithms performances are similar but one of the objectives of this research work is to minimize the number of tasks miss deadline at higher system load. When the system load is less than 1.0, no task misses its deadline in both EFA and NFSA as depicted in figure 4.11 to figure 4.13, but as soon as the load greater than 1.0 the proposed NFSA minimised the deadline miss.

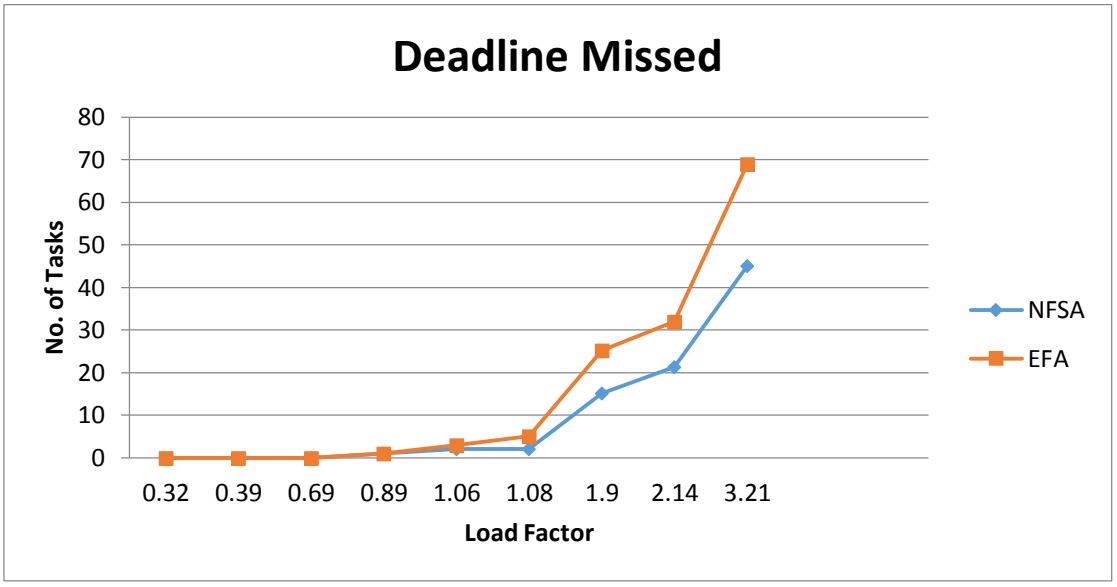


Figure 4.11: Number of Tasks Missed Deadline on 3 Processors

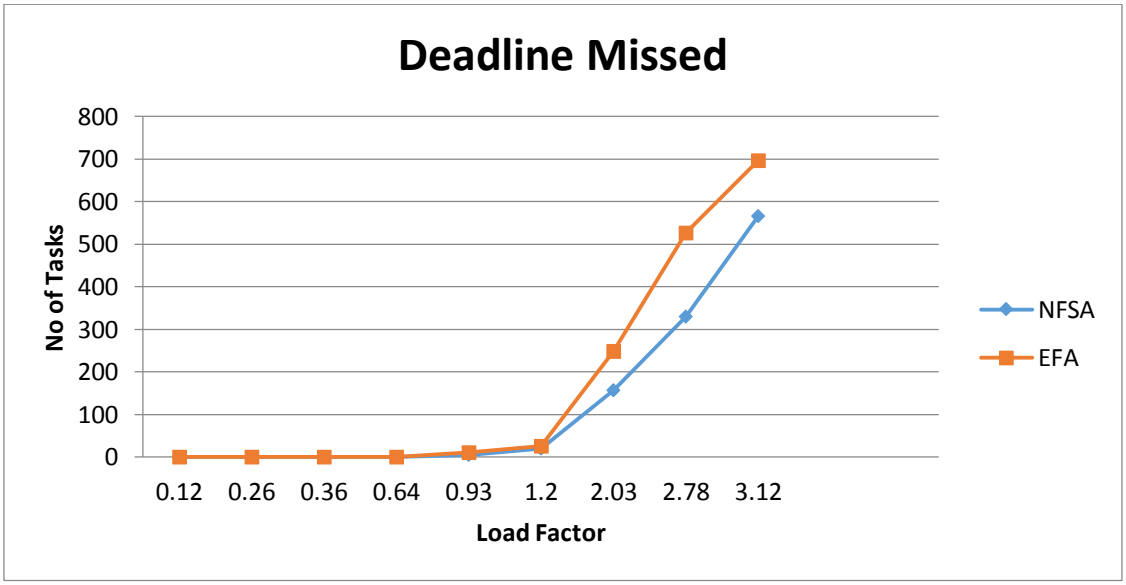


Figure 4.12: Number of Tasks Missed Deadline on 10 Processors

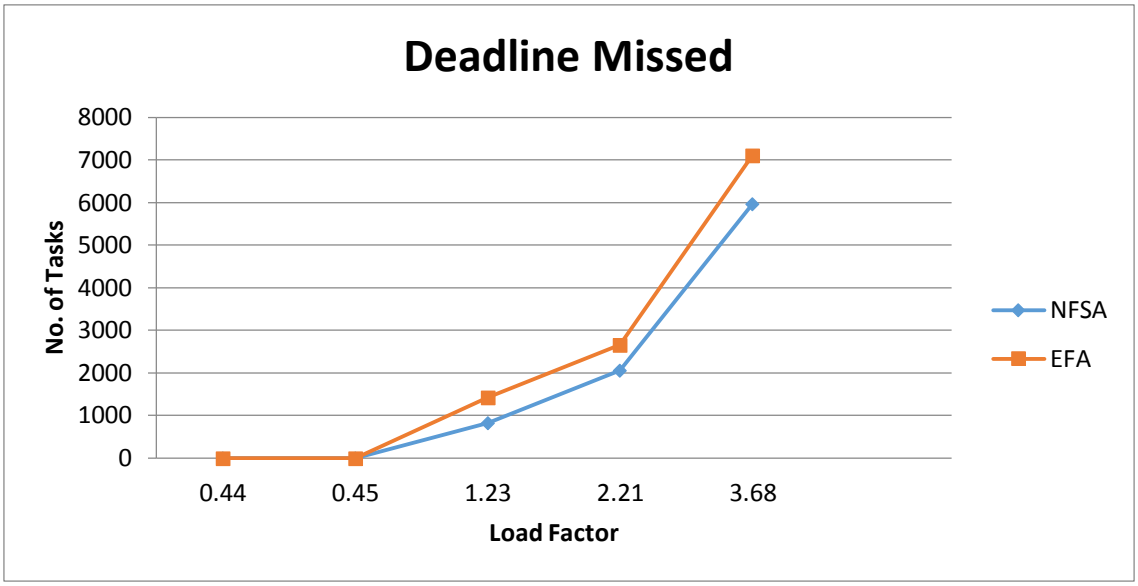


Figure 4.13: Number of Tasks Missed Deadline on 100 Processors

However, the graphical results depicted in figure 4.11 to figure 4.13 has justified our proposed algorithm NFSA that it actually minimized the deadline missed on all the processors at higher system loads.

4.5 PERFORMANCE CRITERIA ANALYSIS TABLE

The following tables in this section depict the summary of performance evaluation of both algorithms. The table 4.7a to 4.7c show the summary of total average turnaround time, total average response time and total number of deadline missed for both EFA and NFSA algorithm on 3, 10 and 100 processors respectively. The results show that NFSA have better performance in these criteria.

Table 4.7a: Average Turnaround Time.

Algorithms	**3 Processors**	**10 Processors**	**100 Processors**
EFA	24.0891	121.452	257.8680
NFSA	16.7778	89.1540	210.3790

Table 4.7b: Average Response Time.

Algorithms	**3 Processors**	**10 Processors**	**100 Processors**
EFA	15	167	2,237
NFSA	10	120	1,769

Table 4.7c: Total Number of Deadline Missed.

Algorithms	**3 Processors**	**10 Processors**	**100 Processors**
EFA	15	167	2,237
NFSA	10	120	1,769

CHAPTER FIVE

SUMMARY, CONCLUSION AND FUTURE WORKS

Real-Time scheduling remain the key to achieving good performance of most systems but most of this systems are embedded into another systems thereby the users have little or no understanding of their functionality as such the choice of choosing the type that best suit problems cannot be over emphasized. Therefore, for a system to select its scheduling algorithms, the users most take into consideration the nature of the tasks, and the required parameters such as the priority, computation time, arrival time and deadline of the tasks. However, real-time systems involved varieties of tasks with deadline thereby missing deadline will result to system degradation in performance of soft real-time tasks.

5.1 SUMMARY

The objectives of this work was to proposed an algorithm NFSA which minimise the average turnaround time, average response time and reduces number of tasks that misses their deadline at higher system load. This model was developed and implemented using java programming language where the user interacts with the system by supplying the desired number of tasks and processors with different ranges of computation time to be simulated. Different test cases where considered and the results were stored in an external files for analysis. However, the results have shown that the proposed algorithm can compete alongside with the existing algorithm EFA in the cases where the system is over loaded.

5.2 CONCLUSION

This research work have finally solved the problem statement stated in chapter one, which is the problem of fair or minimising real-time scheduling on multiprocessors in EFA. The

proposed algorithm NFSA has successfully mapped real-time tasks to processors thereby achieving its stated objectives. Tasks parameters were randomly generated using poisson, uniform and normal distribution. The fuzzy rules were carefully defined in order to capture every possible scenario of real life situations and deadline miss problem in EFA was reduced. Users and designers of operating system will find NFSA helpful in implementing real-time tasks scheduling for some dedicated systems. Therefore, NFSA will be a great option for fair scheduling and minimise the response time, turnaround time and deadline miss of tasks in the real life system.

5.3 FUTURE WORKS

As regards to this research work, the following future works are required;

i. Adaptive method is required to improve the performance of the fuzzy inference engine.

ii. For the time complexity of the system, rule reduction techniques are suggested.

iii. The model design and implementation can be improved on in order to support higher number of tasks and processors.

REFERENCES

Ascia G., Catania V., Ficili G. and Panno D. (2001). A Fuzzy Buffer Management Scheme for ATM and IP Networks, *Institute of Electrical and Electronics Engineers Transactions on Computer Engineering* (IEEE) INFOCOM, Anchorage, 1539-1547, Alaska.

Bashir A. (2013). Fuzzy Round Robin CPU Scheduling Algorithm. *Journal of Computer Science*, doi:10.3844/jcssp.2013.1079.1085, 1079-1085.

Behera H., Ratikanta P., Priyabrata M. (2012). An Improved Fuzzy-Based CPU Scheduling (IFCS) Algorithm for Real Time Systems. *International Journal of Soft Computing and Engineering (IJSCE)*, *2*(1), ISSN: 2231-2307.

Bindi C. (2013). Fuzzy Logic Membership Function. Retrieved March 13, 2014, from http://www.bindichen.co.uk/post/AI/fuzzy-inference-membership-function.html

Bogdan C., Wong C., Jason N. and Clifford S. (2005). Group ratio round-robin: Proportional share scheduling for uniprocessor and multiprocessor systems. In *Proceedings of the USENIX Annual Technical Conference*, 337–352, Anaheim, California.

Bovet D. and Marco C. (2005).Understanding the Linux Kernel. O'Reilly Media, Inc., 3rd edition. 266-274, ISBN:0-596-00565-2.

Buttazzo G., Lipari G., Caccamo, M. and Abeni L. (2002). Elastic scheduling for flexible workload management. *Institute of Electrical and Electronics Engineers Transactions*. *51*(3):289-302.

Carpenter J., Funk S., Holman P., Srinivasan A., Anderson J. and Baruah S. (2004). A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms. Handbook of Scheduling: Algorithms, Models, and Performance Analysis, Published by Chapman Hall/CRC Press, 311-319, Boca Raton, Florida, USA.

Cottet F., Delacroix J. and Mammeri Z. (2002). Scheduling in Real-Time Systems. Published by John Wiley and Sons Ltd, Chichester, West Sussex, England, ISBN:0-470-84766-2.

Dertouzos M. and Mok A. (1989). Multiprocessor On-Line Scheduling of Hard Real-Time Tasks. *Institute of Electrical and Electronic Engineers (IEEE) Transactions on Software Engineering*, *15*(12):132-137.

Fredrik L., Thomas O. and Jessica W. (2010). Scheduling Algorithms for Rea-Time Systems. Retrieved from http://www.idt.mdh.se/kurser/ct3340/ht10/MINIKONFERENCE-SCHEDULE-2010.htm

Galvin P., Abrahm S. and Gagne G. (2004). *Operating System Concepts* (7th ed.). John Wiley and Sons Inc, 12-15, NJ, USA.

Goossens J. and Richard P. (2004). Overview of real-time scheduling problems. Euro Workshop on Project Management and Scheduling.

Klir G. and Yuan B. (1995). Fuzzy Sets and Fuzzy Logic: Theory and Applications. Prentice Hall, *4*(1):1-4, Pretoria.

Liu C. and Layland J. (1973). Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for Computing Machinery* (ACM), *20* (1): 46–61,doi: 10.1145/321738.321743

Mahdi H., Sied M. and Caro L. (2007). Soft Real-Time Fuzzy Task Scheduling for Multiprocessor Systems. *International Journal of Intelligent Technology*. *2*(4): ISSN 1305-6417

Mamdani E. and Assilian S. (1975). An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies*. *7*(1):1-13.

Pedrycz W. and Gomide F. (2007). Fuzzy Systems Engineering: Toward Human-centric computing. Wiley-Institute of Electrical and Electronic Engineers.

Ramamritham K. and Stankovic J. A. (1994). Scheduling Algorithms And Operating Systems Support For Real-Time Systems. Institute of Electrical and Electronic Conference, *82*(1):55-67. Jordan.

Robin G. (2012). Dynamic Scheduling in Real Time for Resource Optimization. (MSc. Thesis. Electronics and Communication Engineering Department, Thapar University, Partiala)**.**

Sabeghi M., Naghibzadeh M. and Taghavi T. (2005). A Fuzzy Algorithm for Scheduling Soft Periodic Tasks in Preemptive Real Time Systems". *International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering* (CISSE), 467-471.

Sabeghi M., Naghibzadeh M. and Taghavi T. (2006). Scheduling nonpreemptive periodic tasks in soft realtime systems using fuzzy inference. 9th *Institute of Electrical and Electronic Engineers.* International Symposium, ISBN:0-7695-2561-X, doi:10.1109/ISORC.2006.70, Korea.

Sagar G., Neha A. and Kamal D. (2012). A Fuzzy Approach For Task Scheduling in a Real Time Distributed System. *International Journal of Research in Engineering and Applied Sciences. 2*(2): 1740-1742, ISSN: 2249-3905.

Shahzad B. and Afzal M. (2006). Optimized Solution to Shortest Job First by Eliminating the Starvation. *The 6th Jordanian International Electrical and Electronics Engineers Conference. Jordan*.

Shatha J. and Kasim A. (2010). Design and Evaluation of a Fuzzy-Based CPU Scheduling Algorithm. Information processing and Management: 45-52, Springer-*Verlag International Journal*, Berlin.

Sheo D., Payal G. and Kawaljeet K. (2012). A Fuzzy Approach Scheduling on More Than One Processor System in Real Time Environment. *International Journal of Scientific Research Engineering & Technology* (IJSRET), *1*(5):289-293, ISSN 2278 – 0882.

Stallings W. (2004). Operating Systems Internals and Design Principles. Prentice-Hall,5th edition. ISBN:0-13-147954-7, Englewood Cliffs.

Strosnider J., Lehoczky J. and Sha L. (1995). The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments. *Institute of Electrical and Electronic Engineers*, Transactions on Computers, *44*(1).

Tanenbaum S. and Woodfhull S. (2005). Operating Systems Design and Implementation, 2nd edition, Prentice Hall, 473-475.

Thai N. (2002). Real-time scheduling in distributed systems. *Parallel Computing in Electrical Engineering, International Conference*, Warsaw, Poland, 165- 170.

William G. (2011). An Optimization Approach to Employee Scheduling Using Fuzzy Logic. (MSc. Thesis, California Polytechnic State University, San Luis Obispo).

Zadeh A. (1965). Fuzzy Sets. *Journal of Information and Control. Vol 8*,338-353.

APPENDICES

Appendix I: The 27 Fuzzy Rules of the proposed NFSA

S.No.	Arrival Time	Computation Time	Deadline	Runtime Priority
1.	Early	Short	Critical	High
2.	Early	Short	Normal	High
3.	Early	Short	Sufficient	Normal
4.	Early	Medium	Critical	High
5.	Early	Medium	Normal	Normal
6.	Early	Medium	Sufficient	Low
7.	Early	Long	Critical	Normal
8.	Early	Long	Normal	Normal
9.	Early	Long	Sufficient	Low
10.	Intermediate	Short	Critical	High
11.	Intermediate	Short	Normal	High
12.	Intermediate	Short	Sufficient	Normal
13.	Intermediate	Medium	Critical	High
14.	Intermediate	Medium	Normal	Normal
15.	Intermediate	Medium	Sufficient	Low
16.	Intermediate	Long	Critical	Normal
17.	Intermediate	Long	Normal	Normal
18.	Intermediate	Long	Sufficient	Low
19.	Late	Short	Critical	High
20.	Late	Short	Normal	Normal
21.	Late	Short	Sufficient	Normal
22.	Late	Medium	Critical	Normal
23.	Late	Medium	Normal	Low

24.	Late	Medium	Sufficient	Low
25.	Late	Long	Critical	Low
26.	Late	Long	Normal	Low
27.	Late	Long	Sufficient	Low

Appendix II: Results of the Simulation in Data.txt

Average Turnaround Time for 3 processors		
Load Factor	**NFSA**	**EFA**
0.32	7.04	8.20
0.39	6.516	8.658
0.69	10.58	12.12
0.89	11.386	14.414
1.06	12.00	15.95
1.08	20.408	27.7806
1.9	33.872	46.4684
2.14	38.994	52.998
3.21	53.78	74.968

Average Response Time for 3 processors		
Load Factor	**NFSA**	**EFA**
0.32	2.80	2.52
0.39	1.788	3.972
0.69	4.86	6.40
0.89	6.64	9.67
1.06	7.70	11.65
1.08	15.292	22.668
1.9	28.672	41.484
2.14	33.856	47.858
3.21	49.392	70.580

Number of Deadline Missed for 3 processors		
Load Factor	**NFSA**	**EFA**
0.32	0	0
0.39	0	0
0.69	0	0
0.89	1	1
1.06	2	3
1.08	2	5
1.9	15	25
2.14	21	32
3.21	45	69

Average Turnaround Time for 10 processors		
Load Factor	**NFSA**	**EFA**
0.12	3.42	4.12
0.26	6.28	7.90
0.36	7.52	9.98
0.64	12.63	16.20
0.93	16.63	23.13
1.2	56.39	77.33
2.03	89.02	126.13
2.78	142.23	200.90
2.12	195.63	228.57

Average Response Time for 10 processors		
Load Factor	**NFSA**	**EFA**
0.12	0.77	1.47
0.26	3.18	4.81
0.36	4.74	7.19
0.64	9.63	13.20
0.93	14.69	20.21
1.2	48.97	69.91
2.03	84.05	121.16
2.78	137.26	195.94
2.12	191.14	224.07

Number of Deadline Missed for 10 processors		
Load Factor	**NFSA**	**EFA**
0.12	0	0
0.26	0	0
0.36	0	0
0.64	0	0
0.93	5	11
1.2	20	26
2.03	156	248
2.78	330	526
2.12	565	696

Average Turnaround Time for 100 processors		
Load Factor	**NFSA**	**EFA**
0.44	66.77	80.381
0.45	56.178	76.92
1.23	143.77	203.18
2.21	304.56	370.48
3.68	529.70	607.48

Average Response Time for 100 processors		
Load Factor	**NFSA**	**EFA**
0.44	52.21	65.81
0.45	53.185	73.92
1.23	138.75	198.16
2.21	290.08	356.01
3.68	517.67	595.44

Number of Deadline Missed in 100 processors		
Load Factor	**NFSA**	**EFA**
0.44	0	0
0.45	0	0
1.23	828	1417
2.21	2049	2656
3.68	5966	7112

Appendix III: Java Source Codes for NFSA

1. Main.java

```
package fuzzy;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.PrintStream;
import java.util.Random;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextField;
/**
 *
 * @author Matthew
 */
public class Main extends JFrame implements ActionListener {
  double total=0;
   JPanel pan = new JPanel();
      JPanel panel = new JPanel();
    JPanel pan1 = new JPanel();
    JPanel pan2 = new JPanel();
    JPanel up = new JPanel();
    JPanel g1  = new JPanel();
    JPanel g2  = new JPanel();
    JLabel noofpl= new JLabel("No. of Tasks:");
    JLabel noofcpul= new JLabel("No of Processors:");
    JTextField noofptf = new JTextField("7");
    JTextField noofcputf = new JTextField("3");
     JButton submit = new JButton("Simulate");
     JButton generate = new JButton("Generate Parameters");
     JLabel title= new JLabel("FUZZY SCHEDULING REAL-TIME TASKS ON MULTIPROCESSOR");
     JLabel froml = new JLabel("Computation Time From:");
     JLabel tol = new JLabel("Computation Time To:");
     JTextField fromtf = new JTextField("1");
     JTextField totf = new JTextField("10");
     JButton result = new JButton("Show Results");
     int n,m;
     double bt,at,pr,tat,cs,rt,ind,copy_of_bt;
     int no_process_processor, each_processor_weight;
     int val=0;
```

```
    double[] At;
    double[] Bu;
    double[] Ep;
    double[] Dl;
    double rtime[];
    double tatime[];
    double pri;
    String[] job;
    JPanel pa1 = new JPanel();
JPanel pa = new JPanel();
   public Main(){
    submit.addActionListener(this);
      generate.addActionListener(this);
      result.addActionListener(this);
       up.setLayout(new GridLayout(5,2));
       pan2.setLayout(new GridLayout(2,1));
       title.setForeground(Color.BLUE);
       setLayout(new BorderLayout());
       up.add(noofpl);
       up.add(noofptf);
       up.add(noofcpul);
       up.add(noofcputf);
       up.add(froml);
       up.add(fromtf);
       up.add(tol);
       up.add(totf);
       up.add(generate);
       up.add(submit);
       pan.add(title,BorderLayout.NORTH);
       pan.add(up,BorderLayout.CENTER);
       pan.add(result,BorderLayout.SOUTH);
       getContentPane().add(pan);
       setTitle("New Fuzzy Schedulling Algorithm (NFSA) Simulator");
       setSize(500,400);
        setLocationRelativeTo(null);
       setVisible(true);
       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
    public static void main(String[] args) {
      new Main();
      }
public int generateArrivalTime(int max){
   int val1 = 0;
   Random ran = new Random();
   val1 = ran.nextInt(max);
   return val1+1;
}
   public void actionPerformed(ActionEvent e) {
     n=Integer.parseInt(noofptf.getText());
     m=Integer.parseInt(noofcputf.getText());
     int a = Integer.parseInt(fromtf.getText());
     int b = Integer.parseInt(totf.getText());
```

```
      try{
       if(e.getSource()==submit){
          try{
          System.out.println("No 1 "+n);
           System.out.println("No 1.1 "+m);
       new NewFuzzy(n,m);
        }catch(Exception eee){
             System.out.println("Error due to "+eee.getMessage());
          }
           try{
        new OldFuzzy(n,m);
          }catch(Exception eee){
             System.out.println("Error due to "+eee.getMessage());
          }
        }else if(e.getSource()==generate){
             // Creating Simulation into file
       createSimulationInput(n,a,b);
       }else if(e.getSource()==result){
          JDialog jd = new JDialog();
          JPanel jp = new JPanel();
          jp.setLayout(new GridLayout(getsize()+getsize2(),1));
          try{
             jd.setTitle("Results Analysis of EFA and NFSA");
          BufferedReader br = new BufferedReader(new FileReader("newdata.txt"));
          String read;
          while((read= br.readLine()) != null){
          jp.add(new JLabel(read));
        }
          BufferedReader br1 = new BufferedReader(new FileReader("olddata.txt"));
          String read1;
          while((read1= br1.readLine()) != null){
          jp.add(new JLabel(read1));
        }
          }catch(Exception es){
             System.out.println("error due to: "+es.getMessage());
          }
          //jd.add(new JScrollPane());
          JScrollPane scrollPane = new JScrollPane(jp);
          jd.add(scrollPane);
          jd.pack();
          jd.setVisible(true);
       }
       }catch(Exception ex){
      System.out.println("Error here due to "+ex.getMessage());
       }
     }
public void createSimulationInput(int n,  int a, int b){
   //Writing to External File
      Bu = new double[n];
      At=new double[n];
      Dl=new double[n];
      Ep=new double[n];
```

```
      int arr = 0;
      job = new String[n];
       PrintStream out =null;
        try{
            out = new PrintStream(new FileOutputStream("data.txt"));
            out.println("JOBs \t AT \t CT \t DL \t E-Priority");
            At=RandomNumber.poisson_value(n);
            Bu =RandomNumber.uniform_value(n, a, b);
//             Bu =RandomNumber.gaussian_value(n, a, b);
            Ep =RandomNumber.uniform_value(n, 1, 20);
            Dl =RandomNumber.uniform_value(n, 200, 500);
                 for(int i=0;i<n;i++)
             {
               job[i]="J"+i;
               if (Bu[i]>=Dl[i]){
                  Dl[i]=Bu[i]+Dl[i];
               }
                out.println(job[i]+"\t"+At[i]+" \t "+Bu[i]+" \t "+Dl[i]+" \t "+Ep[i]);
            }
          JOptionPane.showMessageDialog(null, n+" Tasks Scheduling Parameters were
Sucessfully Generated!", "Success", JOptionPane.INFORMATION_MESSAGE);
          out.close();
       }catch(Exception e){
     System.out.println("Error due to "+e.getMessage());
     JOptionPane.showMessageDialog(null, "Error in creating parameters "+e.getMessage(),
"Error", JOptionPane.ERROR_MESSAGE);
       }
}
public int getsize(){
   int n=0;
  try{
          BufferedReader br = new BufferedReader(new FileReader("newdata.txt"));
          StringBuffer text = new StringBuffer();
          String read;
          while((read= br.readLine()) != null){
n++;
          }
          }catch(Exception es){
             System.out.println("error due to: "+es.getMessage());
          }
    return n;
}
public int getsize2(){
   int n=0;
  try{
          BufferedReader br = new BufferedReader(new FileReader("olddata.txt"));
          StringBuffer text = new StringBuffer();
          String read;
          while((read= br.readLine()) != null){
n++;
          }
          }catch(Exception es){
```

```
            System.out.println("error due to: "+es.getMessage());
          }
    return n;
}
}
```

2. Newfuzzy.java

```
public class NewFuzzy extends JDialog{
  double total=0;
    PrintStream out =null;
   JPanel pan = new JPanel();
    JPanel pan1 = new JPanel();
    JPanel pan2 = new JPanel();
    JPanel up = new JPanel();
    JPanel g1  = new JPanel();
    JPanel g2  = new JPanel();
     JPanel pa1 = new JPanel();
 JPanel pa = new JPanel();
 DecimalFormat df = new DecimalFormat("#.##");
     JLabel title= new JLabel("REAL-TIME TASK SCHEDULING BY NFSA");
    int n,m;
    double bt,at,pr,tat,cs,rt,ind,copy_of_bt;
    int no_process_processor, each_processor_weight;
    int arr = 1;
    int val=0;
  double[] At;
  double[] Bu;
  double[] Ep;
  double[] Dl;
  double rtime[];
  double tatime[];
     double pri;
     double[] nfa;
     String[] job;
    public NewFuzzy(int n, int m){
        try{
            out = new PrintStream(new FileOutputStream("newdata.txt"));
         }catch(Exception e){
         }
       pan.setLayout(new BorderLayout());
        pan2.setLayout(new BorderLayout());
        title.setForeground(Color.BLUE);
        setLayout(new BorderLayout());
        pan.add(title,BorderLayout.NORTH);
        pan.add(new JLabel("New Fuzzy Scheduling Algorithm(NFSA)"),BorderLayout.CENTER);
        out.println("NFSA SCHEDULER");

out.println("=================================================================
=");
        pan.add(new
JLabel("================================================================="),B
orderLayout.SOUTH);
        getContentPane().add(pan,BorderLayout.NORTH);
```

```
       setTitle("Fuzzy Schedulling");
      nalgorithm(n,m);
      setSize(800,700);
       //pack();
       getContentPane().validate();
       getContentPane().repaint();
        setLocationRelativeTo(null);
       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
public void multiple_processor(int n,int pro)
 {
   try{
 pan1.removeAll();
 pan2.removeAll();
 g1.removeAll();
 g2.removeAll();
  pan1.setLayout(new GridLayout(n+1,1));
JLabel subtitle = new JLabel("DISPATCHING JOBS TO PROCESSORS");
out.println();
out.println();
out.println("DISPATCHING JOBS TO PROCESSORS");
  subtitle.setForeground(Color.BLUE);
  pan1.add(subtitle);
  int sum=0;
multiple_processor op[] = new multiple_processor[n+1];      // creatng array for
multiprocessors objects
multiple_processor proc[] = new multiple_processor[pro+1];
rtime = new double[n];
tatime = new double[n];
int []each_processor_weight = new int [pro +1];
int []no_processes_processor = new int [pro +1];
for(int i=0;i<=n;i++)
   op[i]= new multiple_processor();
for(int i=0;i<=pro;i++)
   proc[i]= new multiple_processor();
   for(int j=1;j<pro+1;j++){
      proc[j].no_process_processor=0;
      proc[j].each_processor_weight=0;
      proc[j].ind=j;
   }
for(int m=1;m<=n;m++)
{
op[m].at=At[m-1];
op[m].bt=Bu[m-1];
op[m].pri=nfa[m-1];
}
for(int m=0;m<n;m++)
{
op[m].pr=m;
sum+=op[m].bt;
op[m].copy_of_bt=op[m].bt;
}
```

```
int t=0, processor_index=0,tempo=0,index=0,counter=0;
double tempe=0.0, temp=0;
for(int i=1;i<n;i++)
   {
   for(int j=1;j<n;j++)
      {
      if(op[j+1].pri>op[j].pri)
         {
         tempe=op[j+1].pri;
         op[j+1].pri=op[j].pri;
         op[j].pri=tempe;
         temp=op[j+1].at;
         op[j+1].at=op[j].at;
         op[j].at=temp;
         temp=op[j+1].bt;
         op[j+1].bt=op[j].bt;
         op[j].bt=temp;
         temp=op[j+1].pr;
         op[j+1].pr=op[j].pr;
         op[j].pr=temp;
         }
   }
}
while(t<=sum)
{
   if (counter==0){
      for(int j=1;j<=pro;j++)
       {
       index=j;
       if(op[j].bt>0)
         {
            t+=op[j].bt;
            each_processor_weight[j]+=op[j].bt;
            pan1.add(new JLabel("processor["+j+ "]'s weight = "+each_processor_weight[j]));
            out.println("processor["+j+ "]'s weight = "+each_processor_weight[j]);
            op[j].bt=0;
            no_processes_processor[j]+=1;
            op[j].tat=each_processor_weight[j];
            op[j].rt=op[j].tat-op[j].copy_of_bt;
         }
   }
   counter = counter +1;
}
else{
   for(int j=(pro+1);j<=n;j++)
   {
      index=minimal(each_processor_weight,pro);
         if(op[j].bt>0)
         {
          t+=op[j].bt;
            each_processor_weight[index]+=op[j].bt;
         System.out.println("processor["+index+ "]'s weight = "+each_processor_weight[index]);
```

```
            out.println("processor["+index+ "]'s weight = "+each_processor_weight[index]);
     pan1.add(new JLabel("processor["+index+ "]'s weight = "+each_processor_weight[index]));
            no_processes_processor[index]+=1;
            op[j].tat=each_processor_weight[index];
            op[j].rt=op[j].tat-op[j].bt;
          }
     }
    }
}

JLabel labfcfs = new JLabel("::::::::::::::::::::::::::::EFA RESULT
SIMULATION::::::::::::::::::::::::::::::::");
       labfcfs.setForeground(Color.BLUE);
       pan2.add(labfcfs,BorderLayout.NORTH);
       g1.setLayout(new GridLayout(n+1,9));
       g1.add(new JLabel("TASKs:"));
       g1.add(new JLabel("Arrival Time :"));
       g1.add(new JLabel(" Computation Time:"));
       g1.add(new JLabel("Deadline Time:"));
       g1.add(new JLabel("E-Priority Time :"));
       g1.add(new JLabel("NFA :"));
       g1.add(new JLabel("Response Time:"));
       g1.add(new JLabel("Turn Around Time:"));
       g1.add(new JLabel("Deadline Missed?"));
 int tot_turn=0,tot_res=0;
 for(int i=1;i<=n;i++)
    {
    tot_turn+=op[i].tat;
    tot_res+=op[i].rt;
    }
 float avg_turn=(float)tot_turn/n;
 float avg_res=(float)tot_res/n;
 for(int m=0;m<n;m++)
 {
   System.out.println(" "+job[m]+" \t "+At[m]+"\t\t\t\t "+Bu[m]+"\t\t\t "+Dl[m]+" \t\t\t
"+Ep[m]+" \t\t\t     "+nfa[m]+" \t\t\t     "+op[m+1].rt+" \t\t\t     "+op[m+1].tat+" \t\t\t
"+missedDeacdline(op[m+1].tat,Dl[m]));
    g1.add(new JLabel(job[m]));
    g1.add(new JLabel(""+At[m]));
    g1.add(new JLabel(""+Bu[m]));
    g1.add(new JLabel(""+Dl[m]));
    g1.add(new JLabel(""+Ep[m]));
    g1.add(new JLabel(""+df.format(nfa[m])));
    g1.add(new JLabel(""+op[m+1].rt));
    g1.add(new JLabel(""+op[m+1].tat));
    g1.add(new JLabel(""+missedDeacdline(op[m+1].tat,Dl[m])));
    total = total+((double)Bu[m]/Dl[m]);
    if(op[m+1].tat>Dl[m]){
          val=val+1;
     }
 }
```

```
g2.setLayout(new GridLayout(6,2));
g2.add(new JLabel("Total TAT :"));
g2.add(new JLabel(" "+tot_turn));
g2.add(new JLabel("Total RT :"));
g2.add(new JLabel(" "+tot_res));
g2.add(new JLabel("Avg TAT :"));
g2.add(new JLabel(" "+avg_turn));
g2.add(new JLabel("Avg RT :"));
g2.add(new JLabel(" "+avg_res));
g2.add(new JLabel("Total Deadline Missed:"));
g2.add(new JLabel(" "+val));
g2.add(new JLabel("Load Factor :"));
System.out.println("Total "+total);
out.println("Total TAT : "+tot_turn);
out.println("Total RT : "+tot_res);
out.println("Avg. TAT : "+avg_turn);
out.println("Avg. RT : "+avg_res);
out.println("Total Deadline Missed : "+val);
out.println("Load Factor : "+total*100+"%");
g2.add(new JLabel(" "+total*100+"%"));
pan2.add(g1,BorderLayout.CENTER);
pan2.add(g2,BorderLayout.SOUTH);
 pan1.validate();
 pan1.repaint();
  pan2.validate();
 pan2.repaint();
  pa.add(pan2);
 pa1.add(pan1);
final JScrollPane scrollp = new JScrollPane(pa, JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
final JScrollPane scrollp1 = new JScrollPane (pa1,JScrollPane.VERTICAL_SCROLLBAR_AS_
NEEDED, JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
 getContentPane().add(scrollp1,BorderLayout.WEST);
getContentPane().add(scrollp,BorderLayout.SOUTH);
scrollp.validate();
scrollp.repaint();
scrollp1.validate();
scrollp1.repaint();
    }catch(Exception e){
        System.out.println("Error in multi_processor due to "+e.getMessage());
    }
}
public String missedDeacdline(double tat,double dl){
   String val="N";
   if(tat>dl){
      val="Y";
   }
   return val;
}
public void nalgorithm(int n, int m){
   try{
      //Writing to External File
```

```
      Bu = new double[n];
      At=new double[n];
      Dl=new double[n];
      Ep=new double[n];
      nfa= new double[n];
      job = new String[n];
 double temp,temp1,temp2,temp3;
       double temp4;
       String job1;
       String fileName = "tipper.fcl"; //Reading the fuzzy rules from this file
                                          FIS fis = FIS.load(fileName, true);
       FunctionBlock functionBlock = fis.getFunctionBlock(null);
          if (fis == null){//error while loading?
                                                System.err.println("can't load file:'"+fileName+"'");
                                                return;
                                                }

     if(n>0 && m>=2){
   //Reading Jobs parameters From External File
        try{
          BufferedReader br = new BufferedReader(new FileReader("data.txt"));
          StringBuffer text = new StringBuffer();
          String read;
          int i=0;
          String arr2[];
          while((read= br.readLine()) != null){
          if(i>0){
          arr2=read.split("\t");
          job[i-1]=arr2[0].trim();
          At[i-1]=Double.parseDouble(arr2[1].trim());
          Bu[i-1]=Double.parseDouble(arr2[2].trim());
          Dl[i-1]= Double.parseDouble(arr2[3].trim());
          Ep[i-1] =Double.parseDouble(arr2[4].trim());
          System.out.println(job[i-1]+": "+At[i-1]+": "+Bu[i-1]+": "+ Dl[i-1]+": "+Ep[i-1])
}
          i++;

          }
          }catch(Exception e){
             System.out.println("error due to: "+e.getMessage());
          }

             for(int i=0;i<n;i++)
                {
                   functionBlock.setVariable("arrival", At[i]);
                   functionBlock.setVariable("computation_time", Bu[i]);
                   functionBlock.setVariable("deadline",Dl[i]);
             //evaluate NFA
                                          functionBlock.evaluate();
                                                nfa[i] =
functionBlock.getVariable("priority").getValue();
              }
```

```
    //Sorting the jobs arrays based on NFA priority

    for(int i=0;i<n-1;i++)
       {
          for(int j=i+1;j<n;j++)
             {
             if(nfa[i]<nfa[j])
                {
                 //sorting jobs
                  job1=job[i];
                  job[i]=job[j];
                  job[j]=job1;
                  //swap Burst Arrival
                  temp=At[i];
                  At[i]=At[j];
                  At[j]=temp;
                   //swap Burst
                   temp1=Bu[i];
                   Bu[i]=Bu[j];
                   Bu[j]=temp1;
                  // Swap Deadline
                   temp2 = Dl[i];
                   Dl[i]=Dl[j];
                   Dl[j]=temp2;
                  // Swap Deadline
                   temp3 = Ep[i];
                   Ep[i]=Ep[j];
                   Ep[j]=temp3;
                  // Swap NFA
                   temp4 = nfa[i];
                   nfa[i]=nfa[j];
                   nfa[j]=temp4;
                   }
          }
    }

   System.out.println("SCHEDULING THE JOBS TO PROCESSORS");

    multiple_processor(n,m);
      }else{
           System.out.println("Atleast one job and two processors");
      }
        }catch(Exception e){
         System.out.println("Error in nalgorithm due to "+e.getMessage());
      }
}
public static int minimal(int [] value, int num){
int min, index=0;
   try{
   min = value[1];
    index=1;
```

```
    for (int i=2; i<=num;i++){
       if(value[i]<min){
          min = value[i];
          index=i;
       }
    }
    }catch(Exception e){
        System.out.println("Error in multi_processor due to "+e.getMessage());
    }
  return index;
}
```