ON INFORMATION THEORETIC ENTROPY OF REAL TIME DISTRIBUTED SYSTEMS

BY

RASHMI SHARMA

Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat, Solan–173234, Himachal Pradesh, INDIA

DECEMBER 2014

ON INFORMATION THEORETIC ENTROPY OF REAL TIME DISTRIBUTED SYSTEMS

Thesis submitted in fulfillment for the requirement of the degree of

DOCTOR OF PHILOSOPHY

by

Rashmi Sharma

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING AND INFORMATION TECHNOLOGY,
JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT, SOLAN–173234, HIMACHAL PRADESH, INDIA

DECEMBER 2014

Special Thanks To
Jaypee University Of Information Technology

ABSTRACT:

Utilization factor is the most significant feature of Real Time (RTS) and Distributed System (DS). In a DS if any task requires migration or duplication, first scheduler checks the processor utilization and then relocate/duplicate it towards the destination processor. Likewise, in RTS the acceptance test of every scheduling algorithm is determined by using utilization factor ($Processor\ Utilization \leq 1$). Earliest Deadline First (EDF), Rate Monotonic Scheduling (RMS), Least slack time first, Pfair are some renowned scheduling algorithms of RTS. They usually schedule real time tasks on the basis of utilization imposed by task upon respective processor. However, the unexpected arrival (migrated & duplicate tasks) and execution of tasks create bafflement in the system. This bafflement generates overloading which causes tasks to start missing the deadline. This bafflement is nothing but uncertainty present in the system; if we quantify the amount of uncertainty, then this uncertain task overload can be regulated precisely and efficiently. This amount of uncertainty can be quantified by using the proposed information theoretic concept of entropy in RTDS. Hence, the primary focus of this thesis is arrival of entropy in RTDS as a new guidance parameter. Along with the introduction of this novel parameter, we also analyze new task duplication and migration based scheduling algorithms for DS (tasks having no deadline) and RTDS.

Let us track the path of this thesis briefly, how utilization based dynamics governing algorithm has been simulated and ultimately ended in to entropy-based methodology. Thesis in the beginning proposes duplication based scheduling algorithm (TDASLM) then to remove its drawback of overloading, a new migration based scheduling algorithm for DS has been projected. Afterwards, author move towards RTDS where maintenance of task in a global queue follows the Rate Monotonic strategy. Tasks are assigned to randomly selected processors and threshold based EDF algorithm is used for the execution of tasks. Hence, joint EDF-RM scheduling algorithm is henceforth proposed. From the simulation results, it is observed that the success ratio, maximum tardiness and average CPU utilization give encouraging results as compared to some existing algorithms (EDF, RMS and D_R_EDF).

In above two proposed scheduling algorithms overloading is common. We have designed algorithms to overcome the trouble of overloading in distributed system or missing a deadline in RTDS. Therefore, we have to think about the core of this problem, the main cause here is allocation of tasks and then incapable execution of the same. This powerless behavior of system is because of the uncertainty of processor in tasks admission. Mathematically, this amount of uncertainty can be calculated in terms of entropy. Hence, we have calculated entropy (uncertainty) values of the processor in per-unit time parallel to utilization factor. With our simulations for comparing utilization and information theoretic entropy, the resultant graph of these values with respect to time surprisingly ends up showing one to one mapping between them. Consequently, the focal point has been shifted from utilization factor to the entropy component. From these encouraging results, we thought of replacing utilization factor with entropy. Maximum entropy model (MEM) is applied to determine the upper boundary limit of processor entropy. We also justify the modeled simulation with a mathematical explanation of MEM in RTDS. With the help of interdisciplinary approach, we theoretically describe a new dynamics-governing stricture with some critical advantages for task scheduling of RTDS.

Conclusively, the thesis emphasizes on how and why to use entropy to safeguard RTDS from an overloading problem. This method is unique in two senses, firstly it claims state of art application of information theoretic entropy to RTDS, secondly propounds novel dynamics governing parameter entropy in the same.

ACKNOWLEDGEMENTS

Writing acknowledgement for the people who were engaged in any kind of support in my research work is a very emotional issue for me. Past three and a half years these faces daily came into my sights, partially or fully but their constant encouragement was really substantive. It is impossible to put down all the names here. However, I shall try to put forward those who cannot be forgotten at the time of completion of my research work leading to my Doctorate degree in Computer Science.

I would like to go with my heart-felt gratitude to my supervisor, **Associate Prof. (Dr.) Nitin**, Department of Computer Science and Engineering, for his valuable guidance, support and encouragement throughout my research work. I am indebted to him for his valuable advice, constructive criticism and never-ending patience at each footstep of my research work.

I owe my thanks to **Prof. (Dr.) Satya Prakash Ghrera,** Head, Department of CSE and ICT, **Prof. (Dr.) Harinder Singh,** Head, Department of Mathematics, **Prof. (Dr.) Deepak Dahiya, Dr. Nitin Rakesh** and other DPMC (Doctoral Program Monitoring Committee) members for providing me assistance, moral support, valuable suggestions and necessary facilities during the course of my research work. **Prof. (Dr.) P. B. Barman**, Head, Department of Physics was the person who had a faith in my idea of using entropy for the first time in RTDS. I cannot drop his name in thanksgiving list, I am grateful to your encouragement sir. I too would wish to thank Mr. Amit Kumar Shrivastava and Mrs. Anshul Sood for providing me all software and research work related facilities.

I also wish my appreciation to **Ex-Chief Operating Officer - Jaypee Education System Dr. Y. Medury**, **Vice Chancellor Prof. Shiban Kishen Kak, Director Brig. (Retd.) Balbir Singh** and **Dean (Academic and Research) Prof. Dr. T. S. Lamba** for their interest, encouragement and help during the course of my research tenure at JUIT. I would wish to thank the authorities of JUIT, for furnishing the financial support during my research study. I also like to thank Dr. Shri Ram, Dy. Learning Resource Manager JUIT, for their help and cooperation.

I would wish to thank my friends Atul Saurabh and Shivi Gandhi who helped me technically during my research work. Thanks are due to my dear friends Abhilasha Chauhan, Manoj Gaur, Piyush Chauhan, Priyam Dhani, Sakshi Khanna and Ved Prakash Bhardwaj for their moral support and encouragement during these years of research work.

Finally yet importantly a special thank to my beloved husband Tapan Pancholi without whom I cannot yet think about the starting of PhD. His sacrifice, never ending motivational support, unconditional and constant backing always encouraging me for this life changing experience. This was his aspiration that is coming to reality.

I am forever thankful to my respected Mother, Father, Mother-in-law, Father-in-law, valued elder Brother and loving younger Sister beyond words without their sustained moral support, it would have been impossible to succeed in writing this wonderful thesis.

I shall thank the almighty lord Shiva for blessing me with this insight and inspiration in the subject, when I was low he gave me the courage and patience to surmount the hurdles of physical and emotional in nature. I shall hand out this thesis to lord Shiva first.

(Rashmi Sharma)

Date: 20, December 2014

DECLARATION

I hereby declare that the work reported in the Ph.D. thesis entitled **"On Information Theoretic Entropy of Real Time Distributed Systems"** submitted at **Jaypee University of Information Technology, Waknaghat, India,** is an authentic record of my work carried out under the supervision of Associate Professor Dr. Nitin. I have not submitted this work elsewhere for any other degree or diploma.

(Signature of the Scholar)

Rashmi Sharma

Department Of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, India

Date (20, December 2014)

Date: 20, December 2014

CERTIFICATE

This is to certify that the work reported in the Ph.D. thesis entitled "**On Information Theoretic Entropy Of Real Time Distributed Systems**", submitted by **Rashmi Sharma** at **Jaypee University of Information Technology, Waknaghat**, **India,** is a bonafide record of her original work carried out under my supervision. This work has not been submitted elsewhere for any other degree or diploma.

(Signature of Supervisor)

Dr. Nitin
Associate Professor
Department Of Computer Science & Engineering and Information Technology

Date (20, December 2014)

LIST OF ABBREVIATIONS

Symbol	*Definition*
$\mathrm{C}_{\mathrm{t_i,t_j}}$	Communication cost b/w $\mathrm{t_i}$ and $\mathrm{t_j}$
$\mathrm{C_i}$	Worst Case Execution Time
$\mathrm{D_{x,y}}$	Data transfer rate from processor $\mathrm{p_x}$ to $\mathrm{p_y}$
$\mathrm{S_y}$	Start-up cost on given processor
$\mathrm{t_i}$	Task
A_i	Arrival Time
$Available_{entropy}$	Available entropy of given processor/system
$Average_{task_{meet}}$	Average number of tasks meet the deadline
$Average_{task_{miss}}$	Average number of tasks miss the deadline
B_Q	Boundary limit of global task queue
$Current_{entropy}$	Current entropy of given processor/system
D_i	Task Deadline
$E_{processor}$	Entropy of given processor
$E\rho_{utilization}$	Efficient processor utilization
P_i	Inter-arrival Period
T_i	Set of i independent real time tasks
$tard_{\tau_i}$	Tardiness of given task
$t_{i\ meet}$	Deadline meeting information of i^{th} task
$t_{i\ miss}$	Deadline missing information of i^{th} task
u_{t_i}	Per task utilization
$u_i(t_j)$	j_{th} Task utilization of i_{th} Processor
ρ_{UB}	Processor Upper Bound
ρ_i	i^{th} Processor
τ	Task
$\rho_{utilization}$	Per processor utilization
$\tau_{arrival}$	Task arrival time
τ_{dline}	Task deadline
$\tau_{p,a}$	Task a of p processor
τ_{period}	Inter-arrival period of task
$\tau_{priority}$	Task priority

$\tau_{utilization}$	Per task utilization
$\mathrm{C}(\mathrm{t_i})$	Average computation cost of given task
$\mathrm{CC}(\mathrm{t_{i,k}})$	Computation cost of task $\mathrm{t_i}$ on $\mathrm{k^{th}}$ processor
DAG	Directed Acyclic Graph
DP	Destination Processor
E	Edges of DAG
ET	Execution Time
G	DAG
p	Total number of processors
$\mathrm{pred}(\mathrm{T_i})$	Predecessor of task $\mathrm{T_i}$
RTDS	Real Time Distributed System
RTS	Real Time System
T	Set of tasks
TFT	Total Finish Time
TST	Total time of Scheduling
V	Vertices of DAG
$\mathrm{v}(\mathrm{t_i}, \mathrm{t_j})$	Data sent from task $\mathrm{t_i}$ to $\mathrm{t_j}$.
$A\_preemption$	Average number of preemptions
$CC[t_j][Pi]$	2D Array or Matrix containing Computation
DS	Distributed System
$Entropy(t_i)$	Entropy of i^{th} task
$Entropy(system)$	Entropy of a given system
$Entropy(task_{meet})$	Entropy of meeting deadline.
$Entropy(task_{miss})$	Entropy of missing deadline.
FR	Failure Ratio
$I(E_i)$	Information of i^{th} event
MR	Migration Ratio
$P[]$	Array of Processors of System
SR	Success Ratio
T	Task-set
U	Processor Utilization
$maxent(processor)$	Maximum entropy of given processor
$maxent(task)$	Maximum entropy of given task
$probability(t_i)$	Probability of i^{th} task
$probability(E)$	Probability of each event
$probability(meet)$	Deadline meeting probability of given task
$probability(miss)$	Deadline missing probability of given task

$t[j]$	Array of tasks of System
$tard_{t_i}$	Tardiness of i^{th} task
$u[i]$	Array of utilization of i_{th}processors

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION AND MOTIVATION

The focus of this thesis is a better load balancing among the processors of Real Time Distributed System. There are three main approaches exist in distributed system, i.e. Task assignment, load balancing and load sharing. This thesis deals with load-balancing approach of distributed system from various phases. Task migration and duplication are two very famed methodologies that help to balance the load in-between processors. Beginning portion deals and details some novel methods in this domain. Unconvinced quest remains as what factor informs the scheduler to judge upon task migration or duplication? The answer of this question is utilization factor. What happen if we can replace the utilization by some other parameter for possible improvements? In the later sections, this event has been claimed. Irrefutably this replacement becomes the primary appeal of this thesis.

1.1 *Problem Statement and Contributions*

From the very beginning of the work, many research papers were found reporting task duplication and migration algorithms that are working on tightly coupled distributed systems (multiprocessors) [A. Burchard, 1995; J. Anderson, 2005; P. Chaudhuri, 2010 & N.W. Fisher, 2007]. In addition, very few of them worked out on loosely coupled systems. Thus, we have reorganized these algorithms and proposed a new task duplication based algorithms, i.e. Task Duplication Assisted Schedule Length Minimization (TDASLM) Algorithm. This algorithm is simulated for a loosely coupled distributed system that follows fully connected (mesh) topology. Moreover, bottom up tracing of directed acyclic graph (DAG) has been used. During the simulation of task duplication algorithm we came to know about the problem of overloading that occurred when the scheduler duplicate any task to destination processor. In order to resolve this overloading problem task migration methodology has been implemented. In these proposed algorithms (duplication and migration), utilization of processor is the only factor that decides the destination processors for victim tasks.

Following these above two methodologies, the research work has been published:

Rashmi Sharma and Nitin, *Duplication with Task Assignment in Mesh Distributed System Scheduling*, Proceedings of the IEEE World Congress on Information and Communication Technologies, Mumbai, INDIA, December 11-14, 2011, pp. 672-676.

&

Rashmi Sharma and Nitin, *Optimal Method for Migration of Tasks with Duplication*, Proceedings of the 14th IEEE International conference on Computer Modeling and Simulation (IEEE UKSIM), Emmanuel College, Cambridge, UK, March 28-30, 2012, pp. 510-515.

&

Rashmi Sharma and Nitin, *Duplication with task assignment in Mesh Distributed System,* Journal of Information Processing Systems (JIPS), Vol.10, No.2, pp.193-214, June 2014.

Further, this distributed system was extended to handle the real time tasks, thus real time scheduling algorithms have been enacted with migration methodology. In Real Time System (RTS) Earliest Deadline First (EDF), Rate Monotonic Scheduling (RMS), Static Cyclic Scheduling (SCS), Deadline Monotonic Scheduling (DMS), Least Laxity First (LLF), Pfair Scheduling Algorithm etc. are some frequent algorithms for scheduling real time tasks. Among these algorithms, EDF and RMS are the root of all algorithms [J. Anderson, 2008; J. Anderson, 2005 & G. C. Buttazzo, 2003]. During a literature study of both algorithms, we came to know about following bottlenecks:

1. *In EDF algorithm, because of overloading problem (utilization >1) domino's effect occurs.*
2. *As we know that RMS works on the basis of utilization bound test* $(B(n) = n \times (2^{\frac{1}{n}} - 1))$ *where* n *is a number of tasks.* $n \times (2^{1/n} - 1)$ *is a worst case utilization bound which decreases monotonically from 0.83 when* $n = 2$ *to* $log_e 2 = 0.693$ *as* $n \rightarrow \infty$. *This utilization based result shows that any periodic tasks set will be able to meet all the deadlines when the total utilization is not greater than 0.693 (if RMS is used). Hence, here schedulability of tasks is dependent on following three cases:*
   a) $0 \leq utilization \leq B\ (n)$*: set of* n *Independent tasks is schedulable*
   b) $B(n) \leq utilization \leq 1$*:* n *Independent tasks may or may not be schedulable (no conclusion).*

c) $utilization > 1$: *Occurrence of overloading*

3. *In RMS, requirement is that the deadline and inter-arrival period of tasks should be equal.*
4. *RMS works with static priorities, whereas EDF does on dynamic priorities.*

In decree to surmount all these above-mentioned problems, joint EDF-RM scheduling algorithm is aimed in this dissertation. This proposed algorithm has following properties:

1. *Functioning well in overloading condition (for overloading, utilization threshold is set).*
2. *Deadline of task can be less than or equal to its inter-arrival period.*
3. *This algorithm is dynamic priority based.*

The contribution towards this research work is published as follows:

Rashmi Sharma and Nitin, *Performance Evaluation of New Joint EDF-RM Scheduling Algorithm for Real Time Distributed System*, Journal of Engineering, Hindawi publication Corporation, 2014.
&
Rashmi Sharma and Nitin, *Task Migration with EDF-RM Scheduling Algorithms in Distributed System*, Proceedings of the 2nd IEEE International Conference on Advances in Computing and Communications, Kerala (IEEE ICACC), INDIA, August 9-11, 2012, pp. 182-185.

Till-now every real time scheduling algorithm has worked on the basis of tasks or processor utilization factor [G. Umarani, 2012 & B.T. Akgün, 1996]. It is coarse in all printed papers that utilization is one of the principal parameters that decide the overloading and underloading condition. As we know that utilization refers to a computer's usage of processing resources, or the amount of work handled by a processor to execute the particular task. Hence, we can say that utilization is the percentage of time that the CPU is doing useful work. Moreover, in real time system, the execution of tasks is very uncertain, i.e. dynamically arrival of high priority task can preempt the runnable task due to which some of them miss their deadline. It is possible for the scheduler to compute this amount of uncertainty of task attributes. Thus, based on this computed uncertainty tasks execution rate can be appended. This dissertation suggests a novel component that computes such uncertainty and works based on computed uncertainty values. According to information theory, the amount of disorder present in given information is measured in terms of entropy. We compute the entropy load of particular tasks and overall entropy load of the processor. That intern decides the tasks schedulability parallel to utilization factor. Thus giving us a comparable candidate parameter that competes utilization factor, governing the tasks scheduling dynamics. During our literature survey, we have found some limitations with utilization factor:

1. *Utilization factor is unable to reveal the exact load level of the utilization.*
2. *It is destined to run out in large-scale distributed systems (grid, cloud computing) due to scalability crisis*.*
3. *It has no capability to compute the uncertainty of task execution.*

**Scalability crisis comes to play when the exact load information of available resources is replaced by normalized information. For example, at the time of task migration or scheduling each processor should communicate the exact amount of information about its resource availability, so that scheduler knows how much load (in bits/bytes) can be allocated. However, it refers its utilization that is just a normalized number (0 to 1). For a large DS it is a crisis that slows down the entire system.*

On the other side, when we use entropy in place of utilization factor. The probable advantages are:

1. *Entropy factor communicates exact load information of available resources, so that load accommodation amount is easily quantified.*
2. *Entropy may evade the scalability crisis as it furnishes exact, but not average information on available resources.*
3. *It can predict the amount of uncertainty in task execution as well as guide global task dynamics in a better way.*

The contribution towards this research study is published and is as follows:

Rashmi Sharma and Nitin, *Entropy, a New Dynamics Governing Parameter in Real Time Distributed System: A Simulation Study*, International Journal of Parallel, Emergent and Distributed Systems, Taylor and Francis, Vol. 29, No. 6, pp. 562-586,12 November 2013.

&

Rashmi Sharma and Nitin, *Visualization of Information Theoretic Maximum Entropy Model in Real Time Distributed System*, In 3rd IEEE International Conference on Advances in Computing and Communications (IEEE ICACC), Kerala, INDIA, August 29-31, 2013.

&

Rashmi Sharma and Nitin, *Evaluation and comparison of Load Balancing in RTDS using Information theoretic entropy,* In 4th IEEE International Advance Computing Conference (IACC), pp. 674-679, 2014.

In short, this thesis has worked on a load balancing approach of RTDS with certain aspects. Task generation, execution and migration constitute so much of dynamic activities. In order to control and govern such activities, the system needs a prompt fine-tuning parameter. In any distributed system, this parameter is utilization that guides the dynamics of system. In real time systems, task schedulability is decided by some acceptance test that also works on the basis of utilization. For example, in EDF utilization should be less than or equal to 1 and RMS decides on the basis of its bound test value ($B(n) \leq utilization \leq 1$). At present, this thesis use maximum entropy model [Dong Yu, 2009; W. R. Derek, 2008 & D.Feldman, 2002] that decides the maximum limit of processor's entropy and it replaces the given condition of EDF with $task\ entropy \leq processor\ entropy \leq maxentropy$. To the best of our knowledge, this thesis first time introduces an alternative dynamics governing parameter that deals with space as well as time. This work is the major contribution in RTDS domain.

1.2 *Thesis Outline*

The thesis has been organized in nine chapters out of which CHAPTER 1 presents Introduction. CHAPTER 2 presents groundwork of distributed systems, real time systems, and real time distributed systems along with overview of respective scheduling algorithms. CHAPTER 3 explains the various task duplication terminologies along with proposed task duplication algorithm. This chapter as well explains the mesh topology that is applied for making the interconnection between processors/nodes. CHAPTER 4 explains the drawback and solution of task duplication methodology in distributed systems. In CHAPTER 5, a new real time scheduling algorithm has been discussed in which EDF and RM scheduling algorithms work in concert. Further, CHAPTER 6 introduces the concept of information theoretic entropy in RTDS. Moreover CHAPTER 7 and 8 simulates entropy parameter in real time scheduling algorithm and compares its performance on the basis of various parameters with earliest deadline first and rate monotonic scheduling algorithms. Finally, CHAPTER 9 presents the conclusion of the thesis supported by the result of experiments and simulations followed by the future scope of the research work.

1.3 *Publications*

[1]. Rashmi Sharma and Nitin, Duplication with task assignment in Mesh Distributed System, Journal of Information Processing Systems (JIPS), Vol.10, No.2, pp.193-214, June 2014..

[2]. Rashmi Sharma and Nitin, Performance Evaluation of New Joint EDF-RM Scheduling Algorithm for Real Time Distributed System, Journal of Engineering, Hindawi publication Corporation, Volume 2014 (2014), Article ID 485361, 13 pages, January 2014.

[3]. Rashmi Sharma and Nitin, Entropy, a New Dynamics Governing Parameter in Real Time Distributed System: A Simulation Study, International Journal of Parallel, Emergent and Distributed Systems, Taylor and Francis, Vol. 29, No. 6, pp. 562-586,12 November 2013.

[4]. Rashmi Sharma and Nitin, Evaluation and Comparison of Load Balancing in RTDS using Information Theoretic Entropy, Proceedings of 4th IEEE International Advance Computing Conference (IEEE IACC), ITM University, INDIA, pp. 674-679, February 21-22, 2014.

[5]. Rashmi Sharma and Nitin, Visualization of Information Theoretic Maximum Entropy Model in Real Time Distributed System, Proceedings of the 3rd IEEE International Conference on Advances in Computing and Communications (IEEE ICACC), Kerala, INDIA, August 29-31, 2013, pp. 282-286.

[6]. Rashmi Sharma and Nitin, Task Migration with EDF-RM Scheduling Algorithms in Distributed System, Proceedings of the 2nd IEEE International Conference on Advances

in Computing and Communications, Kerala (IEEE ICACC), INDIA, August 9-11, 2012, pp. 182-185.

[7]. Rashmi Sharma and Nitin, Optimal Method for Migration of Tasks with Duplication, Proceedings of the 14th IEEE International conference on Computer Modeling and Simulation (IEEE UKSIM), Emmanuel College, Cambridge, UK, March 28-30, 2012, pp. 510-515.

[8]. Rashmi Sharma and Nitin, Duplication with Task Assignment in Mesh Distributed System Scheduling, Proceedings of the IEEE World Congress on Information and Communication Technologies, University of Mumbai, INDIA, December 11-14, 2011, pp. 672-676.

CHAPTER 2

BACKGROUND AND PRELIMINARIES

Previous chapter has explained the overview of this thesis. Now we discuss the background of RTDS in detail. RTDS is an incorporation of distributed and real time system or we can say that properties of RTS are used in distributed scenario and vice versa. In a distributed system, CPU utilization is the parameter that helps scheduler to check the load on given processors [S. Dhakal, 2007 & P. Emberson, 2007]. Correspondingly, in real time systems the schedulability test relies upon current processor utilization values. Hence, utilization plays a pivotal role in both systems. As we have mentioned previously that the author has conceptually replaced this utilization value with entropy. The entropy concept comes from the backdrop of information theory propounded by C. E. Shannon [C. E. Shannon, 1949]. Further sections shall explain the background of distributed system, real time system, information theory and entropy.

2.1 *Distributed System*

Distributed System is a collection of autonomous computers/processors that appear to its users as a single articulate system and coordinate with each other in terms of time and resources used. Automated Banking System, Air Traffic Control, Global Positioning Systems, World Wide Web (WWW) etc. are few examples of that. Three ingredients make distributed system: multiple computers, interconnections and shared state between them. Following are some enumerated properties [A. Tanenbaum, 2002]:

1. ***Resource Sharing*:** Distribution and utilization of resources like hardware, software or data from wherever in the systems is resource sharing. Resource manager controls access and concurrency of such resources.
2. ***Concurrency:*** The data are treated with the aid of parallel processing, component accessing and updating in shared resources. If updates of concurrent data are not coordinated then it may violate the integrity of the system.

3. ***Scalability:*** Accommodation of additional computational resources does the amendment in distributed systems due to which processing of particular problem becomes faster. Scalability of the system is achieved by addition of more and faster processors.
4. ***Fault Tolerance:*** The failure of hardware, software and networks comes under the fault of distributed system. The system is somehow maintained in a state where small perturbations may be tolerable to a certain extent.
5. ***Transparency:*** No above-mentioned properties of distributed systems are visible to the users. It covers all distribution of the users as well as the application programs from each other. The functioning of distributed systems is hidden to the clients.

The main characteristic of a distributed system is parallel execution of various independent tasks of a complex problem. It either splits the information to various participant processors or utilizes distributed records to determine the meticulous problem. Many distributed applications make direct use of the programming interface provided by network operating systems. In addition, applications often make use of interfaces to the local file structure. As we have explained, a problem with this loom that allotment is scarcely transparent. A solution is to put an additional layer of software between applications and network operating system, providing a higher degree of generalization. Such a layer accordingly called as middleware. The term middleware suggests that it is software positioned between the operating system and the application as indicated in figure (2.1).

Middleware offers universal services that maintain distributed execution of applications. The middleware conceals the heterogeneity that occurs in a distributed system. This heterogeneity exists in different places:

Programming languages: Different applications can be developed by using different programming languages.
Operating system: It has different individuality and potential.
Computer architectures: Computers have different technical details (e.g., data representations).
Networks: Different network technologies are used to link together Different computers.

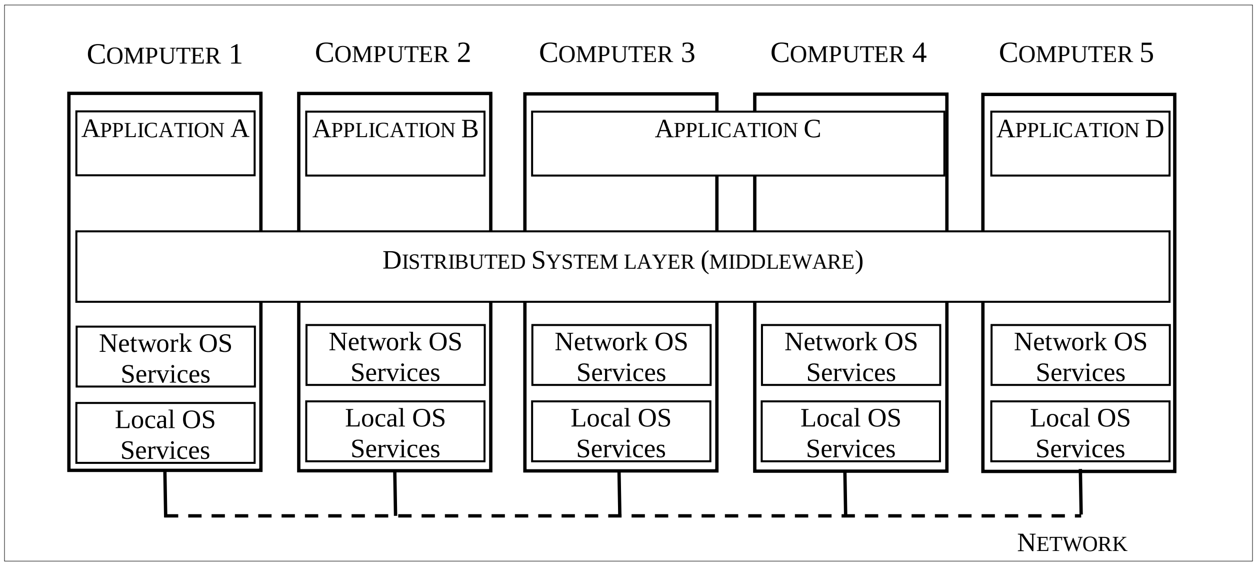


Fig.2.1 General Structure of Distributed System

This is entirely around the architecture and operation of distributed system. As we have mentioned about network heterogeneity, topology of a network also comes under this. Processors of distributed system follow various architectures.

2.1.1 *Topologies:*

With above-mentioned characteristics of distributed system following basic topologies have been evaluated [A.Tanenbaum, 2002 & G.Couloris, 2001]:

Centralized: Centralized systems are well-known form of topology. Client/Server pattern used by databases, web servers are few examples of the centralized distributed system. All tasks and information are centralized into one server through which many clients connecting directly to send and receive the information (figure (2.2 (a))).

Ring: In centralized architecture, server cannot handle high client load. Hence, a common solution is to use a cluster of machines arranged in a ring that act as a distributed server. Communication between the nodes synchronizes state sharing, producing a collection of nodes that offer alike functions, but have failover and load-balancing capabilities (figure (2.2 (b))).

Hierarchical: Hierarchical systems have an entirely different set of advantages from that of rings. Hierarchical organizations are somewhat convenient in that they have a clear chain of activity. However, because these systems have such a broad scope, it can be hard to correct a host with a problem. Hierarchical systems are extensible in a way that any host in the system can add user data or resources, but the conventions of data management may determine how they can be added. The main advantage of these systems is their incredible scalability, i.e. new nodes can be added at any level to reduce too much load (figure (2.2 (c))).

Decentralized: Another topology we consider here is decentralized systems, where all peers communicate symmetrically and have equal responsibilities. Gnutella, Freenet or OceanStore are some decentralized systems. Additionally, the Internet routing architecture itself is largely decentralized where border gateway protocol is used to synchronize the peering links between various autonomous systems (figure (2.2 (d))).

Mesh: In mesh topology, every device is associated with other devices of the system. Such topology has the advantage that if any device or transmission line fails then there are various alternate ways for two nodes to communicate. Simplicity in troubleshooting and increase in reliability are their additional advantages. Internet is the best example of mesh routing. Based on their connectivity mesh topology is divided into following two types:

Full mesh topology occurs when every node/processor has a connection with every node/processor of the system. It is costly to employ, but yields maximum amount of redundancy. Such systems are generally reserved as a backbone network (figure (2.2 (e))).
In ***Partial mesh topology***, some nodes are arranged in a full mesh system, but others are simply connected with one or two nodes in the network. Its implementation is less expensive and yields the minimum amount of redundancy as compared to full mesh topology. It is also known as *a decentralized system* (figure (2.2 (d))).

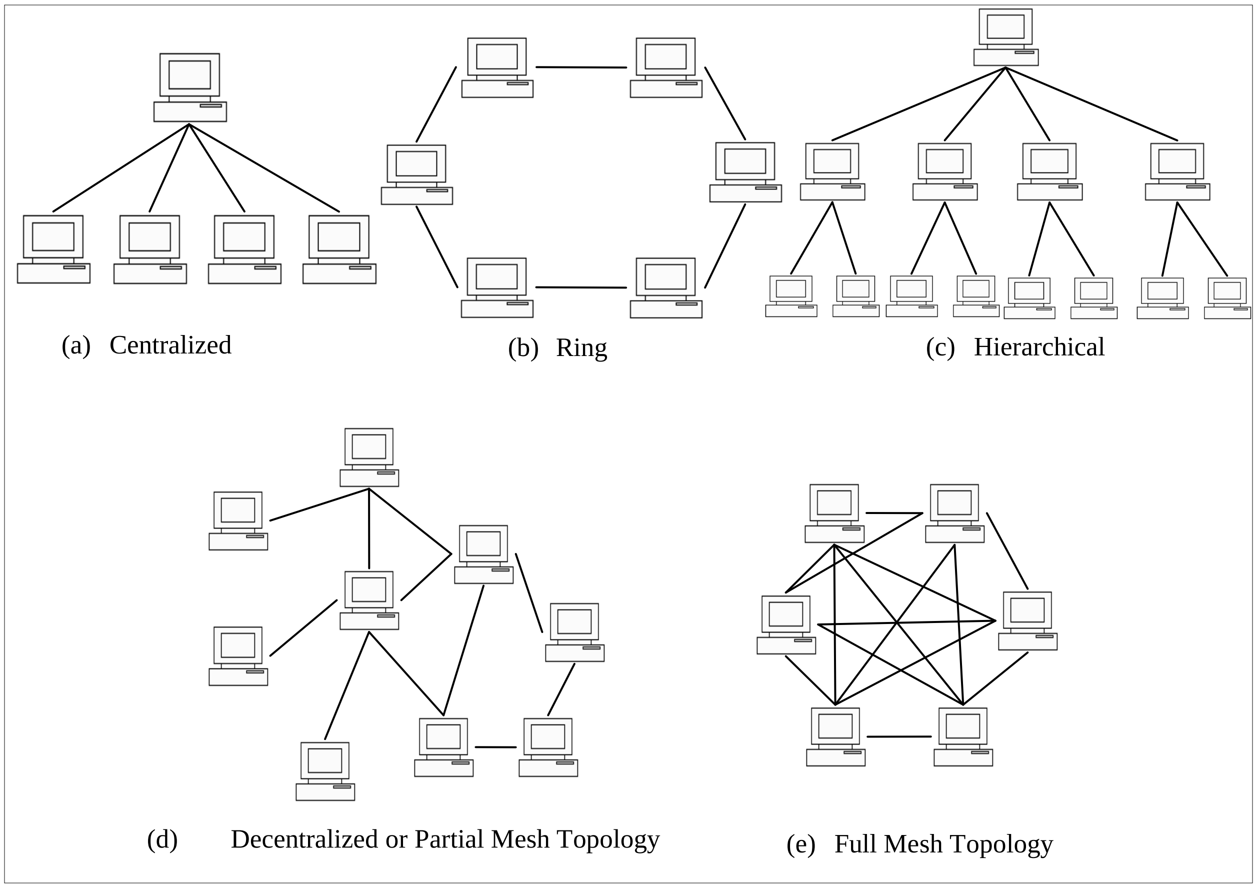


Fig.2.2 Distributed System Topologies

In order to schedule tasks, distributed system schedulers classify into two major approaches: global and partitioning based scheduling (figure (2.3)).

2.1.2 *Global Scheduling:*

In global scheduling, global task queue is maintained where all tasks arrive. The scheduler will assign tasks to processors for execution. Scheduler selects processor dynamically for task allocation and if the processor is not able to execute assigned tasks, then that task will migrate towards another processor. Global scheduler is supported in those systems where average as well as worst-case response time is essential. Thus, in RTDS global scheduler is valuable.

2.1.3 *Partitioning Based Scheduler:*

In partitioned scheduler, tasks are deterministically assigned to processors for execution. If processor is not able to execute the task, then the processor has no right for migration of task to other processors. Migration is prohibited in partitioned scheduler.

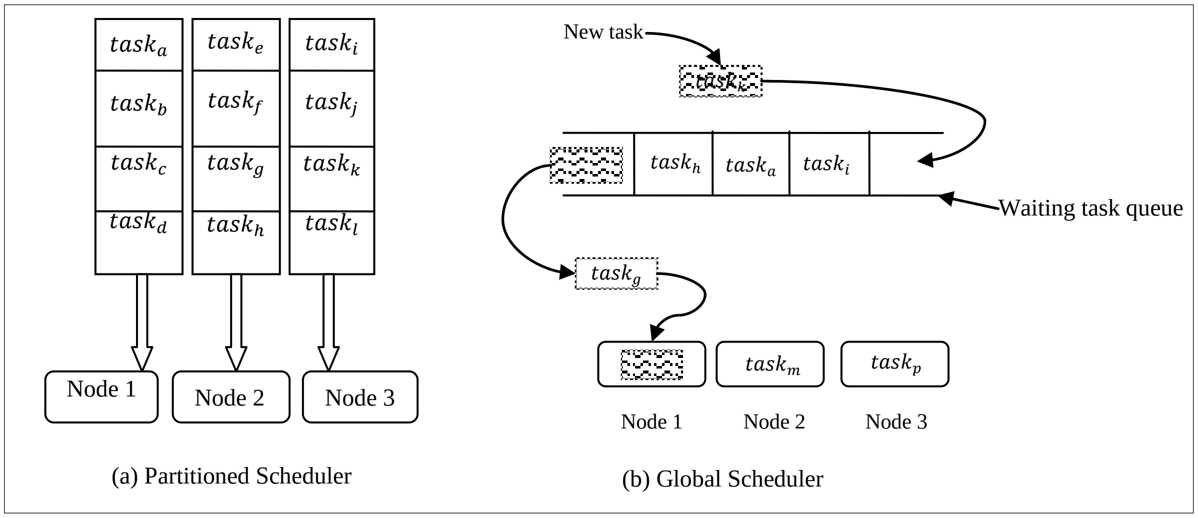


Fig.2.3 Distributed System Schedulers

The main attractive quality of a distributed system is load balancing and parallel task execution. Task migration and duplication are two methodologies that are used to balance the load among processors.

2.1.4 *Task Migration and Duplication*

In order to balance the load of system task of heavily loaded processor migrate towards lightly loaded processor. The task migration technique is used for dependent as well as independent tasks. Here, only original task is executed on assigned processor [H. C. Wang, 2011 & P. Emberson, 2007].

Other methodology used to balance the load is task duplication. In task duplication, the replica of original task is executed on assigned processor. Parallel executions of the same task on various processors take place on task duplication technique [D. Sekhar, 1997; R. Sharma, 2011 & S. Ranaweera, 2000].

Hence, the basic difference between task duplication and migration is following:

Table2.1. Comparison Between Task Duplication and Migration

Task Duplication	Task Migration
Duplicate tasks from one processor (Source) to another processor (Destination).	Migrate tasks from one processor (Source) to another processor (Destination).
Multiple copies of single task execute on multiple processors.	Solo copy of task executes on single processor.
It can increase the problem of overloading on processors.	Helps to reduce the overloading of processors.

Now, next section of this chapter will discuss about RTDS that follows the real time properties.

2.2 *Real Time Distributed System*

A distributed system is a collection of self-governing computers that connects through a network, which facilitate processors to synchronize their actions and resources. The correctness of the system depends not merely on the logical resolution of the computation, but also on the time at which the results are produced [M. Joseph, 1996 & P. A. Laplante, 1993]. In this system, tasks have a timing constraint that is known as deadline of the task.

Following are some temporal parameters of real time tasks:

1. ***Arrival Time:*** It is an instance when the task becomes available for execution.
2. ***Inter-arrival Period:*** Inter-arrival period of any task is the amount of time between the arrival of one task and the arrival of next task.
3. ***Worst Case Execution Time (WCET):*** The worst-case execution time (WCET) of a computational task is the maximum time the task could take to execute on a given hardware platform.
4. ***Deadline:*** It is a strict instance, or assigned time by which task execution is delimited.

Established on such timing constraints RTS is divided into two types. This division of RTS is based on the functional criticality of jobs, usefulness of late results and deterministic or probabilistic nature of the constraints. If a particular task fails to meet its deadline and there is no occurrence of any catastrophe; only the system's overall performance becomes shorter when growingly jobs with soft deadlines complete late, then it comes under a soft RTS. However, if the failure of deadline creates any debacle like system failure or loss of life then it is a hard RTS. This thesis deals with soft RTS.

In real time system, there are two categories of scheduling algorithms: first is static and second is dynamic priority based scheduling algorithms [M. Joseph, 1996 & P. A. Laplante, 1993].

2.2.1 *Dynamic priority based scheduling algorithms*

In dynamic priority scheduling algorithm the priorities to the tasks are assigned during their execution. The objective of dynamic priority scheduling is to acclimatize dynamically changing progress and form an optimal configuration in a self-sustained manner. EDF, Pfair etc. are few illustrations of dynamic priority based scheduling algorithms. As EDF is the basis of most dynamic priority based scheduling algorithms. Thus, we have chosen EDF as the base scheduling algorithm in this dissertation. Let us take an overview of EDF algorithm.

Earliest Deadline First Scheduling Algorithm

EDF is a dynamic priority based scheduling algorithm that works on the basis of: nearer the deadline higher the priority of task [O. Zapata, 2005 & K. Ramamritham, 1990]. It dynamically assigns priority to tasks.

$$\tau_{priority} \propto \frac{1}{\tau_{dline}} \tag{2.1}$$

Before the assignment of priorities to the task, at the very beginning scheduler ensures the schedulability test by examining the task as well as processor's utilization value. Here task sets T is schedulable if

$$\rho_{utilization} \le 1 \qquad (2.2)$$

In order to explain the behavior of EDF in overloading case let us discuss following examples in single as well as multiple processors (Distributed System).

Table2.2. EDF Algorithm in uniprocessor and distributed system as well

EDF scheduling algorithm in uniprocessor	*EDF scheduling algorithm in distributed system*
BEGIN 1. **If** $\tau_{utilization} \le 1$ 2. **If** $\rho_{utilization} \le 1$ 3. The task is schedulable and assign $\tau_{priority}$ of task on given processor 4. **Else task is non-schedulable** END	BEGIN 1. **If** $\tau_{utilization} \le 1$ 2. **If** $\rho_{utilization} \le 1$ 3. **then** task is schedulable and assign $\tau_{priority}$ of task on given processor 4. **Else** migrate the task 5. **Else task is non-schedulable** END

<u>Example of EDF in uniprocessor:</u>

Table2.3. Arrival Time, wcet, Period, Deadline of tasks τ_1, τ_2, τ_3

Tasks	*Estimate Arrival Time*	*Computation time (wcet)*	*Period*	*Deadline*	$\tau_{utilization} = \frac{\tau_{wcet}}{\tau_{Period}}$
τ_1	0	1	3	3	0.33
τ_2	1	2	5	5	0.4
τ_3	3	1.8	4	4	0.45

Figure (2.4) explains the behavior of EDF in uniprocessor case. Three tasks τ_1, τ_2 *and* τ_3 are running on a single processor. Every time arrival of high priority task preempted already running low priority jobs, due to which tasks miss the deadline. Arrival of task τ_3 preempts τ_2 task due to which τ_2 task has missed its deadline. This preemption occurs due to the higher priority of new task τ_3.

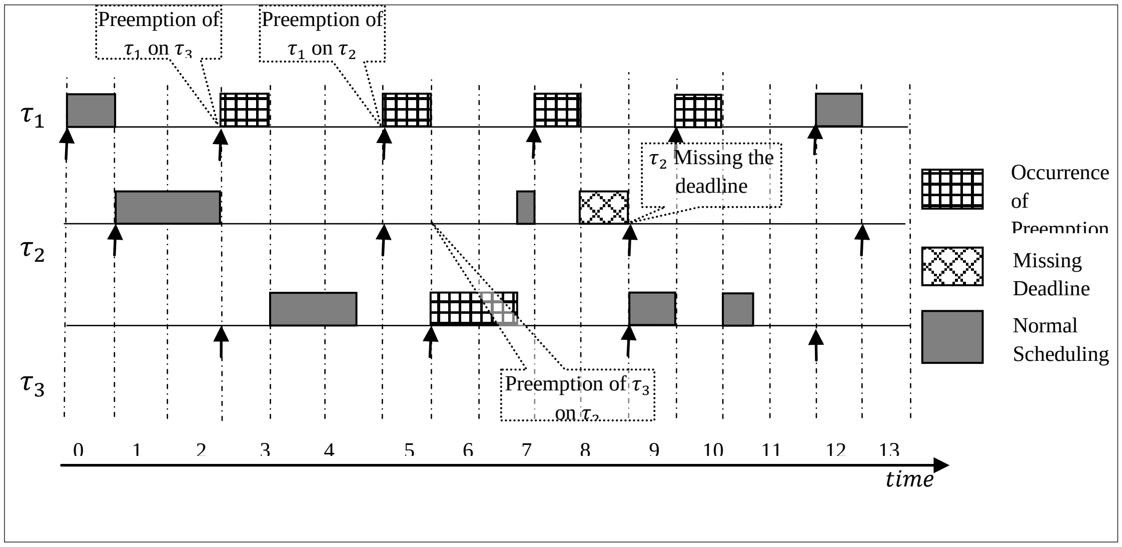

Preemption of τ_1 on τ_3
Preemption of τ_1 on τ_2
τ_1
τ_2 Missing the deadline
τ_2
τ_3
Preemption of τ_3 on τ_2
0 1 2 3 4 5 6 7 8 9 10 11 12 13
time
Occurrence of Preemption
Missing Deadline
Normal Scheduling

Fig.2.4 EDF Scheduling On Single Processor

Example of EDF in Distributed System:

Table2.4. Arrival Time, wcet, Period, Deadline and node for assignment of tasks $\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6$.

Tasks	*Arrival Time*	*Computation time (wcet)*	*Period*	*Deadline*	$\tau_{utilization} = \frac{\tau_{wcet}}{\tau_{Period}}$	*Processor*
τ_1	0	1	3	3	0.33	ρ_1
τ_2	1	2	5	5	0.4	ρ_1
τ_3	3	1.8	4	4	0.45	ρ_1
τ_4	5	3	6	6	0.5	ρ_2
τ_5	1	0.5	2	2	0.25	ρ_3
τ_6	2	2	4	4	0.5	ρ_3

In distributed case, migration technique is employed to overcome the overloading problem of CPU. Few tasks of overloaded processor will migrate towards another processor where utilization is less than one [J. Anderson, 2005].

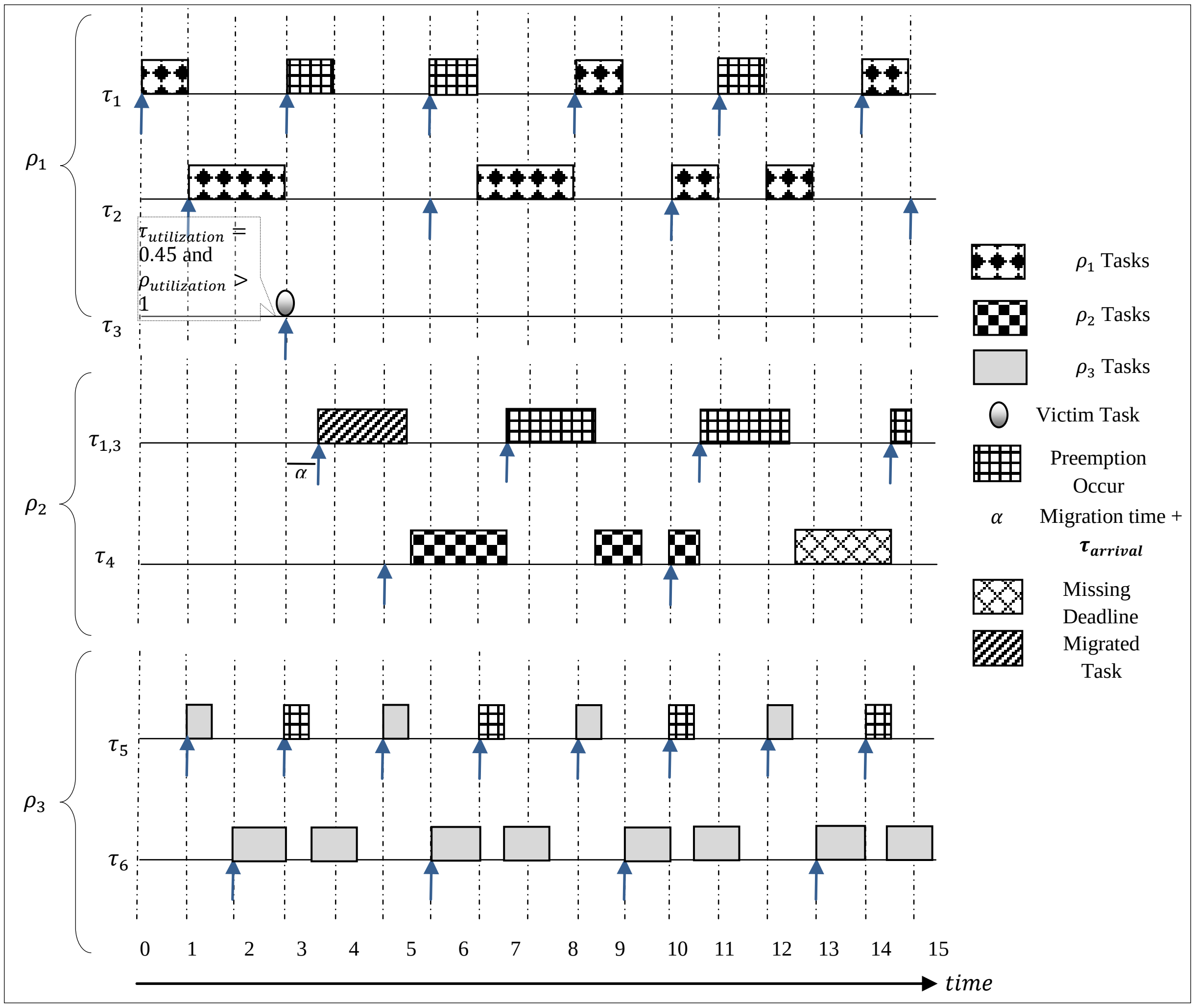


Fig.2.5 EDF scheduling in distributed system

In figure (2.5), utilization of task τ_3 is 0.45 which meet the requirements of scalability, but due to the increasingly arrival of new tasks, ρ_1 utilization becomes 1.15 which is greater than one. The arrival of task τ_3, ρ_1 processor becomes overloaded. Hence, migration method is used here to balance the load. The processor having least utilization value becomes the destination processor for the victim task (task selects for migration). In above example processor ρ_2 utilization is less than 1. During migration when task transfers from one processor to another processor, reallocation of task takes some migration time α. Given that, the migration time is negligible when compared with fundamental unit time of task operation.

2.2.2 *Static priority based scheduling algorithms*

The static priorities are assigned on the basis of cycle duration of the job: shorter the cycle duration is, the higher the job's priority. Rate Monotonic Scheduling (RMS), Static Cyclic Scheduling (SCS), Deadline Monotonic Scheduling (DMS), Least Laxity First (LLF) are few examples of static priority based scheduling algorithms.

As author has mentioned in previous chapter that this thesis will discuss an algorithm in which EDF and RMS algorithms are used together. So, henceforth takes an overview of RMS algorithm.

Rate Monotonic Scheduling (RMS) Algorithm

RMS is a static priority based algorithm and its working is based on the logic: shorter the inter-arrival period higher the priority of task [V. Darera, 2006 & J. Lehoczky, 1989].

$$\tau_{priority} \propto \frac{1}{\tau_{period}} \tag{2.3}$$

For this algorithm, T task-set is schedulable on a given processor if [V. Darera, 2006]

$$\rho_{UB} \leq n(2^{1/n} - 1) \tag{2.4}$$

Where n is total number of tasks

Table2.5. RMS algorithm for uniprocessor and distributed system as well

RMS scheduling algorithm in uniprocessor	*RMS scheduling algorithm in Distributed system*
BEGIN 1. **If** $\tau_{utilization} \leq 1$ 2. **If** $\rho_{UB} \leq n\left(2^{1/n} - 1\right)$ 3. task is schedulable and assign $\tau_{priority}$ of task on given processor 4. **Else task is non-schedulable** END	BEGIN 1. **If** $\tau_{utilization} \leq 1$ 2. **If** $\rho_{UB} \leq n\left(2^{1/n} - 1\right)$ 3. **then** task is schedulable and assign $\tau_{priority}$ of task on given processor 4. **else** migrate the task 5. **Else task is non-schedulable** END

Example of RMS in uniprocessor:

The accomplishment of RMS algorithm is discussed here with the data of a given table (2.6). In RMS algorithm $n \times (2^{1/n} - 1)$ is a worst case utilization bound which decreases monotonically from 0.83 when $n = 2$ to $\log_e 2 = 0.693$ as $n \rightarrow \infty$. This utilization based result shows that any periodic tasks set will be able to meet all the deadlines when the total utilization is not greater than 0.693 [V. Darera, 2006].

$$\mathrm{Lim}_{n\rightarrow\infty} n(\sqrt[n]{2} - 1) = \ln 2 \approx 0.693 \tag{2.5}$$

$$\tau_{1\, utilization} = \frac{\tau_{wcet}}{\tau_{period}} = \frac{1}{3} = 0.33, \tau_{2\, utilization} = 0.4, \tau_{3\, utilization} = 0.45 \tag{2.6}$$

$$\rho_{utilization} = \sum_{i=1}^{2} \tau_{i\, utilization} = 0.33 + 0.4 = 0.73 \tag{2.7}$$

As we remarked before that in RMS worst case utilization bound of processor is 0.693 as $n \rightarrow \infty$. Equation (2.6) mentions the individual utilization values of tasks $\tau_{1,}\ \tau_2\ and\ \tau_3$. In equation (2.7) the cumulative addition of first two tasks $\tau_1\ and\ \tau_2$ is 0.73 that lies between 0.693 and 1, hence both tasks are schedulable. However, arrival of third task increases cumulative utilization to 1.15, which is greater than 1. Therefore, third task may or may not be schedulable on a given processor.

Table2.6. The following table shows some values of ρ_{UB} on different number of tasks n

Number of Tasks (n)	*Tasks*	$\boldsymbol{\tau_{utilization}} = \frac{\boldsymbol{\tau_{wcet}}}{\boldsymbol{\tau_{Period}}}$	$\rho_{utilization} = \sum_{n=1}^{3} \tau_{n\, utilization}$	$\rho_{UB} = n \times (2^{1/n} - 1)$	*Comparison* $\rho_{utilization}$ *with* ρ_{UB}	*Conclusion*
1	$\boldsymbol{\tau_1}$	0.33	0.33	1.0	$0.33 \leq 1.0$	τ_1 is schedulable
2	$\boldsymbol{\tau_2}$	0.4	0.43	0.8284	$0.43 \leq 0.82$	τ_2 is schedulable
3	$\boldsymbol{\tau_3}$	0.45	0.88	0.7797	$0.88 \geq 0.77$	τ_3 can be schedulable

As we know that static priorities are assigned to tasks in RMS. Based on such criteria, priority of tasks in figure (2.6) is in descending order: $\tau_1 > \tau_2 > \tau_3$ and execution of tasks takes place accordingly. Individual utilization values of tasks $\tau_1, \tau_2\ and\ \tau_3$ and processor utilization $\rho_{utilization}$ on every arrival of tasks are mentioned in the table (2.6). In figure (2.6), initially tasks starts with normal scheduling and after that preemption due to higher priority task starts and hence task τ_3 miss its deadline due to preemption.

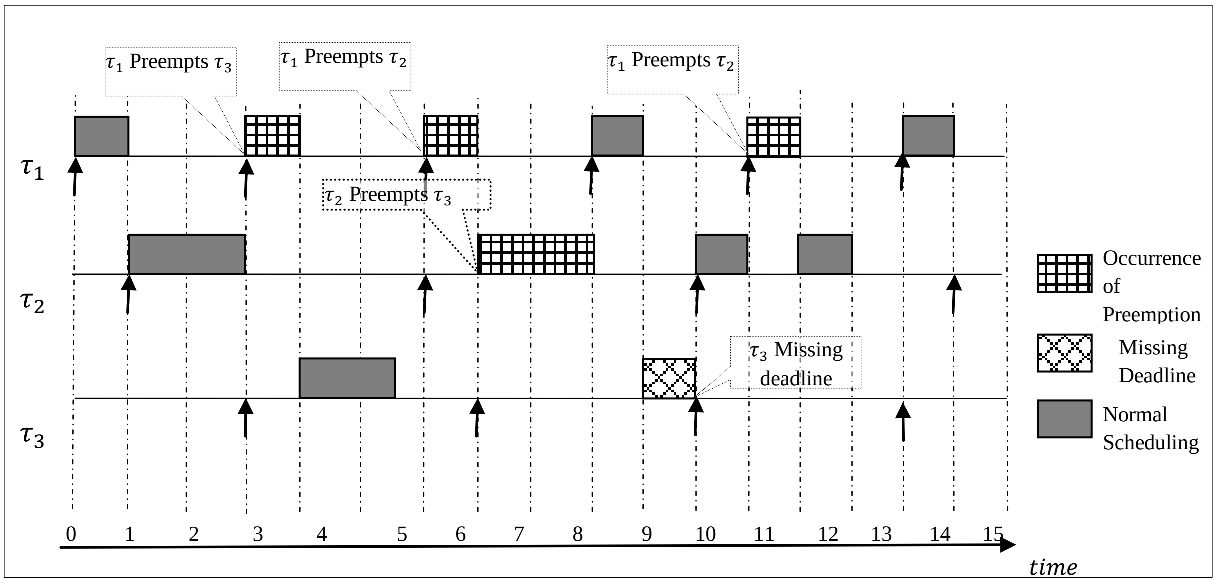

τ_1 Preempts τ_3
τ_1 Preempts τ_2
τ_1 Preempts τ_2
τ_2 Preempts τ_3
τ_3 Missing deadline
τ_1
τ_2
τ_3
Occurrence of Preemption
Missing Deadline
Normal Scheduling
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
$time$

Fig.2.6 RMS Scheduling On Single Processor

<u>Example of RMS in Distributed System:</u>

The computation method of processor utilization $\rho_{utilization}$ is discussed previously in table (2.6). Arrival of third task exceeds the limit of processor utilization. Here, we will discuss how this overloading is managed in distributed systems. In the beginning, ρ_2 processor is redundant and utilization of ρ_3 processor is 0.75. As processor ρ_2 is idle, so it will be a destination processor for overloaded task. Now, victim task τ_3 is migrated towards ρ_2 and its utilization becomes 0.45.

After the simulation of above-mentioned proposed algorithms, authors have noticed that only utilization of a given processor generates dynamism (task generation, migration and execution) in the system. Till now, utilization is the only parameter that plays a vital role in balancing the load as well as scheduling algorithms. After getting some shortcomings of utilization factor in this thesis author has proposed, a new dynamics governing parameter information theoretic entropy for RTDS. The concept of Information theory is proposed together with the US mathematicians C. E. Shannon (1916-2001) and Warren Weaver (1894-1978) in 1949 [C.E. Shannon, 1949], it focuses on how to transmit data most efficiently and economically, and to detect errors in its transmission and reception. After that many researchers has figured out on it and utilized this information theory in several disciplines. Applied mathematics, electrical engineering, Bioinformatics, computer science, statistical inference, natural language processing, cryptography, neurobiology, etc. are some fields where information theory exploits successfully. All above-mentioned fields motivate author to use information theoretic entropy in RTDS. Hence, next section gives the general idea of information theory, entropy and the principle of maximum entropy. Usage/Role of information theoretic entropy in RTDS will be explained in further chapters.

2.3 *Information Theory*

Information theory is a branch of applied mathematics and computer science involving the quantification of information. Claude E. Shannon first developed Information theory to find the fundamental limits on signal processing operations such as compressing data and on reliably storing and communicating data [D. Feldman, 2002].

Following is the general procedure that computes the information on given event E

$$I(E) = -\log_2 P(E) \ or \ \log_2 \frac{1}{P(E)} \tag{2.8}$$

Where $P(E)$ is the probability distribution of an event E. How author has computed this probability distribution and information of an event (real time tasks) of RTDS will be explained in chapters 7 and 8.

2.4 *Information Theoretic Entropy*

A key measure of information is entropy, which is usually expressed by the average number of bits needed to store or communicate one symbol in a message. Entropy quantifies the uncertainty involved in predicting the value of a random variable [D. Feldman, 2002].

The basic method that computes the entropy of given event E is

$$Entropy(E) = P(E) \times I(E) \text{ or } Entropy(E) = P(E) \times \log_2 \frac{1}{P(E)} \qquad (2.9)$$

Let us consider an event E has set of probabilities $P_1, P_2, P_3, \dots\dots\dots, P_k$ then the entropy of this event is

$$Entropy(E) = \sum_{i=1}^{k} P_i \times \log_2 \frac{1}{P_i} \qquad (2.10)$$

2.5 *Principle of Maximum Entropy*

The principle of maximum entropy first expounded by E.T. Jaynes (Edwin Thompson Jaynes) in 1957 [E. T. Jaynes, 1957]. According to the principle of maximum entropy, if nothing is known about a distribution except that it belongs to a certain class, then the distribution with the largest entropy should be chosen as the default. The motivation is twofold: first, maximizing entropy minimizes the amount of prior information built into the distribution; second, many physical systems tend to move towards maximal entropy configurations over time. It specifies that, in order to calculate the accurate verification of previous data, the probability distribution that represents best current state of information is the one that gives largest entropy. In other words, we should choose the probability that gives the maximum entropy value, which decides the maximum extent of entropy values of an event.

In general, the entropy, because it is expressed in terms of probabilities, depends on the observer. One person may have different knowledge of the system from another, and therefore would calculate a different numerical value for entropy. The Principle of Maximum Entropy is used to discover the probability distribution, which leads to the highest value for this uncertainty, thereby assuring that no information is accidentally assumed.

2.6 *Summary*

In this chapter, the background of RTDS along with information theoretic entropy and maximum entropy value has been hashed out. We hereby claim that we have used information theoretic entropy for the first time in RTDS. Any activity in RTDS can be called information processing hence, bound to follow information theoretic concepts. Here entropy comes into play parallel to utilization factor (has been discussed in chapter 6). The maximum entropy value decides the entropy limit of a particular processor / system that helps in scheduling of real time tasks. Thus, guiding the dynamics of RTDS. Further chapters will explain the relation between entropy and utilization or real time distributed system. Additionally, scheduling algorithms for task duplication and task migration with the help of utilization factor and replacement of utilization with entropy parameter will be talked about in further chapters.

CHAPTER 3

DUPLICATION WITH TASK ASSIGNMENT IN MESH TOPOLOGY

Distributed System consists of numerous self-ruling processors that communicate via interconnection network. As previous chapter discussed that, each network follows different connectivity architectures, known as network topology. Mesh topology is one of the topologies [L. N. Bhuyan, 1984] that are employed in this chapter for network connectivity. However, handling of mesh topology is very difficult because of the inter-connectivity between every node. Such network connectivity among processors can be of homogeneous or heterogeneous type. Homogeneous systems share identical architecture, whereas diverse architecture flows in heterogeneous. Therefore, task scheduling is complicated in heterogeneous systems due to non-uniform speed and communication bandwidth. List-based and cluster based are two important scheduling classes that help in task scheduling of heterogeneous systems [D. Bozdag, 2006]. In order to resolve the complication of processor heterogeneity, the author has used cluster based scheduling. Based on processor computational capacity [Y. Jegou, 1997] entire system splits into three clusters (High, Medium and Low).

Parallel task execution is the primary advantage of distributed system. In parallel execution independent subtasks executes correspondingly on various processors. These subtasks are generated from single task that is called as a Directed Acyclic Graph (DAG) that shows interdependency in-between subtasks. In order to accomplish the complete DAG as fast as possible, subtasks are allocated to separate processors on the given system. These processors execute allocated tasks in parallel according to their computational speed. After achieving the results, destination processor transmits it to the source processor (origin) of tasks. This chapter explains the strategic duplication of tasks on the various processors that finally reduces the schedule length of the entire DAG.

Execution of any task passes through following two heuristics:

1. Partitioning heuristic under which tasks split into dependent/independent tasks known as DAG [J. Lopez, 2000]. This DAG represents the size of each task along with computational power consumption.
2. Allotment of processors to these distributed sub-tasks is another phase. First-Fit, Worst-Fit, Best–Fit and Communication aware worst-fit (CAWF) are some task assignment heuristics [J. Lopez, 2000; A. Burchard, 1995 & C. Wang, 2007] that works with/ without task duplication.

These partitioning and assignment heuristics come under the scheduling problem. This problem is also known as grain size determination [J.E.G. Coffman, 1996], clustering problem [A. Bashir, 2013& J. Baxter, 1989] or internalization pre-pass [B. Kruatrachue, 1988].

First-Fit, Worst-Fit and Best-Fit heuristics work in a sequential manner and duplication of task is not followed here. CAWF is designed for the reduction of communication cost in which two dependent tasks (predecessor-successor) can be allocated on a same processor that reduces the communication cost between tasks. In case of multiple successors of single predecessor, CAWF assign one of the successors on the same processor with its predecessor and rest successors use worst case heuristic for allocation. Hence, this is the downside of CAWF algorithm.

This chapter discusses a new task duplication method that will overcome the limitation of CAWF. Author has chosen basic heuristic algorithms (where duplication is not allowed), CAWF and HEFT-TD algorithms to compare with proposed algorithm. Since these algorithms have their own properties, time complexities and advantages during task assignment. There are many other algorithms for the execution of DAG in heterogeneous environment, i.e. DBUS and HEFT-TD [D. Bozdag, 2006 & P. Chaudhuri, 2010] algorithms (few properties are comparable to proposed algorithm with different approach).

In this division, author has projected a duplication of task at the time of its allocation before the execution. In this projected algorithm, DAG is traversed from bottom to up approach that checks the interdependencies of tasks. If two independent tasks are found then those tasks will execute independently (parallel). Next section will discuss about some existing scheduling algorithms of task duplication along with proposed task duplication algorithm followed by its performance.

3.1 *Existing Scheduling Algorithms*

Load balancing is one of the main approaches of distributed system. This load balancing is accomplished by using task duplication or migration in-between processors. As we are dealing with dependent tasks, duplication of tasks is employed here. Main role of task duplication is to reduce the communication cost that helps in diminution of the overall schedule length of entire DAG. Many researchers have suggested various strategies of task duplication [S. Ranaweera, 2000; P. Chaudhuri, 2010 & J. Singh, 2012].

DAG is an arrangement of multiple tasks, out of which some tasks are dependent on previous tasks and some are independent. In case of dependency, successor tasks could not execute before the execution of dependent predecessor tasks. On the other side, independent tasks can execute in parallel on several processors. In a DAG, $G = (V, E)$, E is a link between two nodes that explains the communication cost between two dependent tasks. These subtasks (tasks) are assigned to various processors based on following features already discussed in many other papers [P. Chaudhuri, 2010; R. Sharma, 2011 & S. Ranaweera, 2000]:

Definition3.1: Computation cost (Execution time) of any task on a given processor is dependent on the computational capacity of a particular processor. Time taken by a processor to execute a particular task is known as the computation cost. Computation cost also depends on the size of task as well.

Consider $CC(t_{i,k})$ is the computation cost of task t_i on k^{th} processor from p number of processors. Hence, the average computation cost of any task $CC(t_i)$ is defined as:

$$CC(t_i)=\sum_{k=1}^{p} CC(t_{i,k})/p \qquad (3.1)$$

Definition3.2: Communication Cost ($C_{t_{i,}t_j}$) is the time consumed by the processor in sending data (results) of one task to another processor. This communication cost is dependent on the volume of communicating data and data transfer rate from source to destination processor [P. Chaudhuri, 2010 & J. Singh, 2012].

$$C_{t_{i,}t_j} = S_y + \frac{v(t_i,t_j)}{D_{x,y}} \qquad (3.2)$$

If two jobs are assigned on same processor, then the communication cost, $C_{t_{i,t_j}} = 0$.

Definition3.3: Total Finish Time (TFT) [E.G. Coffman, 1998]: Total finish time of k^{th} tasks on P_n processor is

$$TFT(P_n, k) = \sum_{i=1}^{k}(new\ arrival\ (T_i) + \ Execution\ time(T_i)) \tag{3.3}$$

$$new\ arrival(T_i) = Execution\ time\big(pred(T_i)\big) + C_{pred(T_i),T_i} \tag{3.4}$$

P_n is the number of processors i.e. $P_1, P_2, P_3, \ldots \ldots \ldots P_n$ and k are number of tasks scheduled on given processor. Hence, $TFT(P_n, k)$ is total finish time of k^{th} task on n^{th} processor.

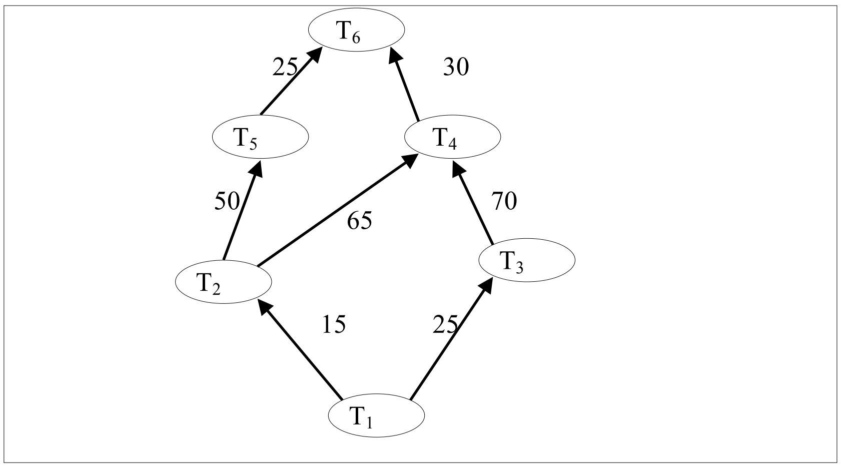


Fig. 3.1. Arbitrary DAG with Communication Cost

Figure (3.1) explains the DAG that contains tasks (subtasks) $\{T_1, T_2, T_3, T_4, T_5, T_6\}$ and $\{25,30,50,65,70,15,25\}$ are their respective communication costs in-between the dependent tasks. Later on, the generation of subtasks (tasks of the DAG) will be assigned to respective processors. Task assignment is the process of multiple task allocation to the numerous processors along-with parallel allocation and execution method for the same [Lo.V.M., 1988].

In distributed system, the selection of processors for task allocation can be sequential or parallel. For sequential task allocation First Fit (FF), Best Fit (BF) and Worst Fit (WF) are well known. All these mentioned sequential allocation heuristics focuses on computation costs but not on communication costs. In [C.Wang, 2007] author has discussed an assignment heuristic approach that focuses on communication cost along with computation cost. This heuristic is known as communication aware worst fit (CAWF). According to CAWF, same processor is assigned to a pair of predecessor-successor that brings down the communication cost in-between assigned pair. However, if one predecessor has multiple successors then the worst fit algorithm is used for rest successors. Although, sequential assignment of tasks is also present here but this algorithm seems helpful in reducing the communication cost.

Equation (3.3) calculates the total finish time of complete DAG on a particular processor. This TFT is dependent on the execution cost of every subtask (task) on the respective processors and communication cost between dependent tasks (subtasks). Table (3.1) explains the execution cost (computation cost) of tasks on respective processors:

Table 3.1. Execution costs of tasks to processors

P_j / T_i	P_1	P_2	P_3	P_4
T_1	35	5	15	10
T_2	9	4	10	7
T_3	6	8	4	12
T_4	23	45	15	26
T_5	10	7	9	11
T_6	30	9	5	18

This section is working on distributed system, parallel execution and allocations of tasks are considered here. Let us consider the case when processors are selected randomly for task assignment and execution as well. In figure (3.2), P_3 processor is selected randomly for T_1, T_4 *and* T_6 tasks; P_2 is assigned to T_3 and T_5; similarly P_1 execute T_2 task. Based on their execution cost on assigned processors and communication costs between tasks overall DAG schedule length has been calculated. In arbitrary selection, DAG schedule length may vary because it is dependent on preferred processor. There is no criterion of processor selection for task execution in arbitrary method.

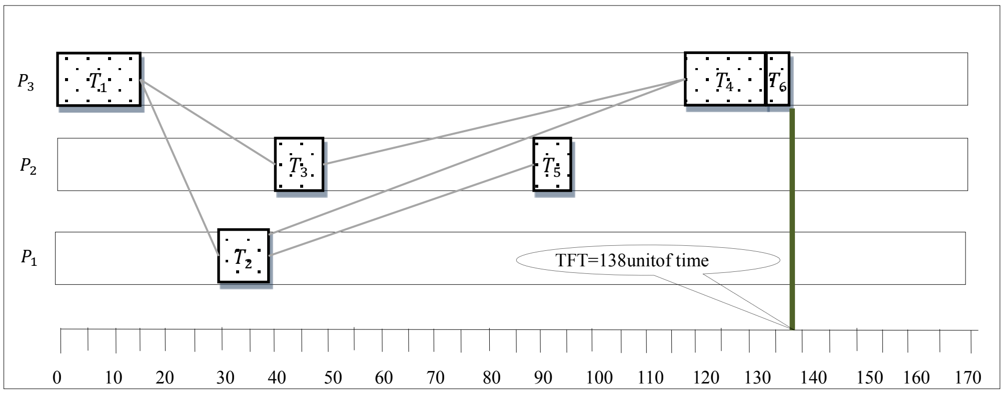


Fig. 3.2. Arbitrary allocation of tasks on processors of distributed system

$$DAG\ Schedule\ Length = \max_{1\le n} TFT(P_n) \tag{3.5}$$

$$= \max\big(TFT(P_1), TFT(P_2), TFT(P_3)\big) = max(39,96,138)$$

$$= 138\ unit\ of\ time \tag{3.6}$$

Now, in figure (3.3) tasks are assigned according to CAWF algorithm. Tasks having predecessor and successors are allocated to the same processor and another task will follow worst fit. (Figure (3.1)) T_1 is the only predecessor of tasks T_2 & T_3. Similarly, T_2 is predecessor of tasks T_4 & T_5. According to CAWF, one of the successors of these predecessors will allocate on the same CPU and other tasks following worst fit. Therefore, T_1 and T_3 (dependent tasks) are assigned on processor P_1. Similarly, T_2, T_4, T_5 and T_6 are interdependent tasks and sequentially assigned to next processor P_2. Lastly, on the basis of computation and communication cost DAG schedule length has been calculated which is lesser than the previous method due to reduction in communication costs.

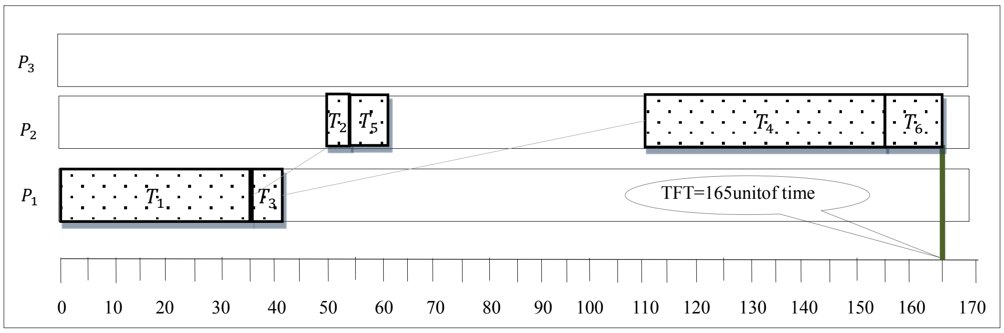


Fig. 3.3. DAG Execution using CAWF heuristics

$$DAG\ Schedule\ Length = max(TFT(P_1), TFT(P_2)) = max(41,165)$$

$$= 165\ unit\ of\ time \qquad (3.7)$$

Further, Third type of allocation is proposed task duplication algorithm that is essentially an advanced adaptation of CAWF. In this method, tasks having less execution cost as compared to communication cost becomes a duplicated task on a given processor.

From above example task T_1 duplicates on P_4 processor because its communication costs towards dependent tasks T_2 and T_3 is greater than its computation cost on particular processors. Similarly, the computation costs of other dependent tasks are greater than their communication costs and therefore those tasks will not duplicate on other processors. By applying such duplication technique, overall schedule length of DAG is comparatively lower than the previous methods.

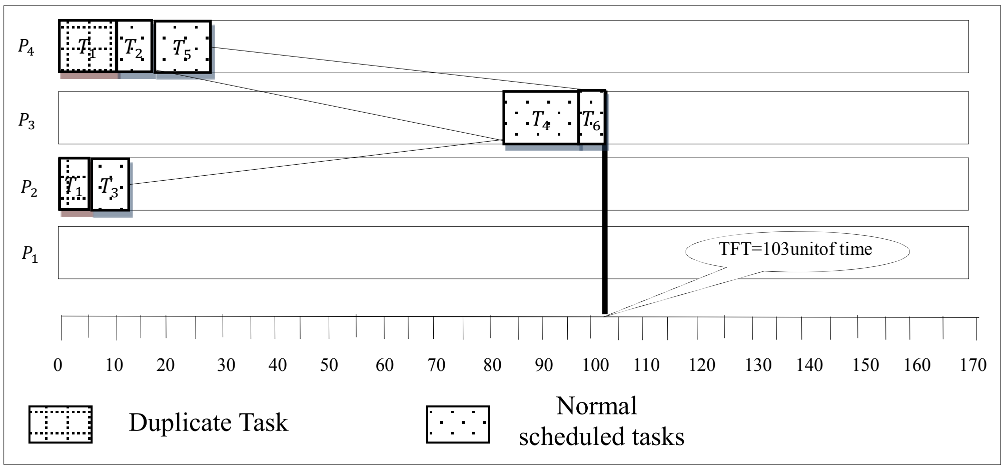


Fig. 3.4. Proposed Task Duplication methodology

$$DAG\ Schedule\ Length = max(TFT(P_2), TFT(P_3), TFT(P_4)) = max(13,103,28)$$

$$= 103\ unit\ of\ time \qquad (3.8)$$

The proposed duplication algorithm is somewhat similar with HEFT-TD and DBUS algorithms. Additionally, the approach used here is different. The approach used in HEFT-TD in top-down whereas bottom-up approach is used in present scenario. Therefore, proposed algorithm gives alike or little bit improved result than the existing. Next section will explain the new algorithm of task duplication followed by simulation results.

3.2 *Task Duplication Assisted Schedule Length Minimization (TDASLM) Algorithm*

There are many approaches has been used for task assignment *i.e.* First-fit, Best-first, Worst-fit and CAWF etc. but all these heuristics, select processors sequentially (first processor assigns first and so forth) for the assignment of tasks without duplication. CAWF algorithm reduces the communication cost by assigning the predecessor and successor on a single processor. This approach works fine if the predecessor has single successor therefore, downside of CAWF approach is multiple successors of single predecessor. As we recognize that, the primary motive of task duplication is to reduce the communication cost that affects the overall schedule length of DAG. Hence, in order to overcome the problem of CAWF task duplication methodology has been used. Now a day's numerous researchers have designed many task duplication algorithms [D.Bozdag, 2006; P.Chaudhuri, 2010 & J. Singh, 2012] with different approaches.

Topology we are using is a mesh that connects every processor with each other processor of the system. After the generation of DAG on given processor, proposed algorithm uses Bottom-up traversing of DAG similar to DBUS algorithm [D.Bozdag, 2006]. This approach determines the dependency and independency in-between sub-tasks of DAG. Independent tasks can execute in parallel and duplication is used for dependent tasks. Task assignments depend on the computational capacity of an assigned processor because the job will execute on allotted processors. Duplication of task is based on the communication cost and execution cost of processors. At the time of duplication, few critical things must remember:

1) Limited number of duplicates: Algorithm must understand the number of duplications of any task (Successor/Predecessor). Avoid useless duplication of tasks. Consider the C_{t_i,t_j}between i^{th} and j^{th} task is less than the $C(P_j)$ of j^{th} processor then there is no need of task duplication.
2) By Bottom-up traversing of DAG all child tasks execute first and then parent task. Due to which parent task duplication decreases.

In the remainder of this section different module of task duplication has elucidated.

3.2.1 *Clustering of Heterogeneous Processors with Mesh Topology*

Mesh topology is used here for the interconnection of heterogeneous processors. Therefore, processor computational power shows a discrepancy. In order to handle this heterogeneous behavior of system, the complete distributed system splits into three clusters (based on computational capacity) i.e. High, Medium and Low. For the grouping of processors, we have fixed some range that determines the efficiency level of processors. These ranges make a decision randomly from 0 to 10.

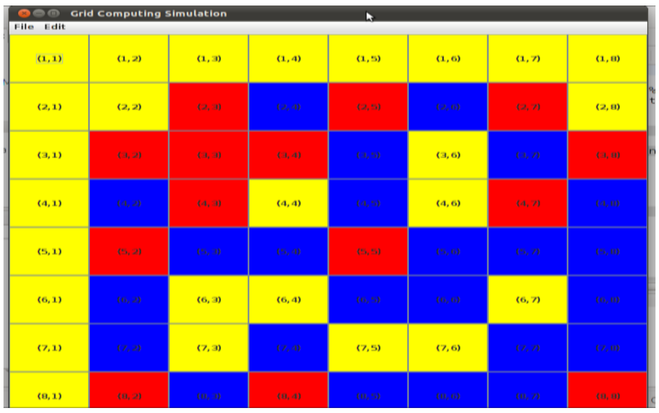
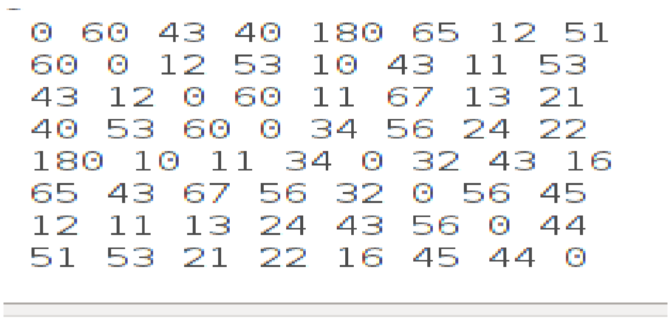


Fig. 3.5. Clustering of Processors

In figure (3.5), each cell represents a node (processor) and based on efficiency range complete system divides into three groups:

Blue color represents "Low efficiency" that comes under 0 to 4 ranges. Yellow color represents "Medium efficiency" and this range lies between 5 and 7. Lastly, Red color is for "High efficiency" and its range lies from 8 to 10.

Along with efficiencies these nodes also have communication costs in-between and author represent this cost with the help of adjacency matrix. Figure (3.6) is a matrix of communication costs between several CPUs. For example: $C_{3,2}$.



```
0 60 43 40 180 65 12 51
60 0 12 53 10 43 11 53
43 12 0 60 11 67 13 21
40 53 60 0 34 56 24 22
180 10 11 34 0 32 43 16
65 43 67 56 32 0 56 45
12 11 13 24 43 56 0 44
51 53 21 22 16 45 44 0
```

Fig. 3.6. Communication costs between nodes

3.2.2 *Generation of Task on Nodes*

In a distributed heterogeneous system, DAG's (tasks) can be generated on any node at any time. In figure (3.7), task generation on particular processor is indicated by green color.

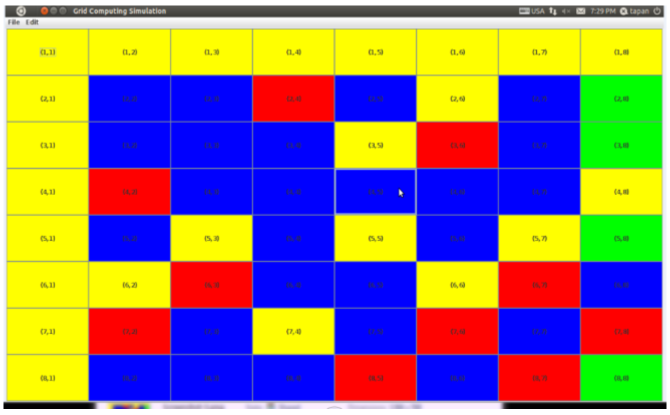


Fig. 3.7. Task Generation on any node of system

Following is the task generation algorithm:

The algorithm generates task randomly on any node and by getEfficiency() function retrieve the efficiency of a particular node.

```
BEGIN
  TASKEXECUTION-ACTIONPERFORMED (java.awt.event.ActionEvent evt)

1. Calendar c= Calendar.getInstance()
2. long m=c.getTimeInMillis()
3. Random r=new Random(m)
4. xcor=r.nextInt()
5. m=c.getTimeInMillis()
6. r.setSeed(m)
7. ycor=r.nextInt()
8. jbArray[Math.abs(xcor%5)][Math.abs(ycor%5)].setBackground(Color.GREEN)
9. group.getEfficiency(Math.abs(xcor%5),Math.abs(ycor %5))

END
```

After the generation of DAG following algorithm retrieve the efficiency of that node and its communication cost with near (other) nodes.

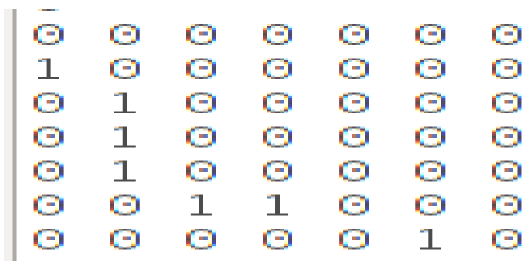
---

getmatrix() function obtain the communication costs from one processor (where task generate) to other nodes. gettaskmatrix() function set the DAG on particular node.

```
SHOW-ACTION-PERFORMED (java.awt.event.ActionEvent evt)
BEGIN
     1. ndag.getmatrix()
     2. tdag.gettaskmatrix()
END
```

3.2.3 *DAG Matrix and its Tracing*

Above module is the basic framework of simulation. This module explains the DAG (in matrix format) of tasks that shows the dependency/independence between tasks. In DAG matrix, 0 represents an independent task and 1 represents a dependency between two tasks (figure (3.8)).



```
0   0   0   0   0   0   0
1   0   0   0   0   0   0
0   1   0   0   0   0   0
0   1   0   0   0   0   0
0   1   0   0   0   0   0
0   0   1   1   0   0   0
0   0   0   0   0   1   0
```

Fig. 3.8. DAG representation in terms of matrix

For the execution of complete DAG Bottom-up approach is used. Task T_7 is independent task (Column of T_7 contains 0), T_6 is dependent on T_7 (T_6 column has 1 on T_7 row). Similarly, other dependencies have been made. For traversing of this matrix of tasks (DAG); first, we check the dependencies (occurrence of 1's in a column) and based on this occurrence sorting of tasks are done. This computation takes $O(n^2)$ time.

Input: A sequence of n subtasks of DAG $(t_1, t_2, t_3 \ldots\ldots\ldots t_{taskDAG})$.
Output: DAG in terms of matrix has been generated.

```
  BEGIN                               cost       times
   1. count=0                         $c_1$      1
   2. for  i=0 to taskDAG.length      $c_2$      $n+1$
   3. for  j=0 to taskDAG.length      $c_3$      $n^2$
   4.     if taskDAG[j][i]==1         $c_4$      $n-1$
   5.        count++                  $c_5$      $n-1$
   6. End for
   7. End for
END
```

Hence, in the worst case, the running time of DAG generation is

$$\begin{aligned} T(n) = {} & c_1.1 + c_2.(n+1) + c_3.n^2 + c_4.(n-1) + c_5.(n-1) \\ & = c_1 + c_2.n + c_2 + c_3.n^2 + c_4.n - c_4 + c_5.n - c_5 \\ & = c_3.n^2 + (c_2 + c_4 + c_5)n + (c_1 + c_2 - c_4 - c_5) = O(n^2). \end{aligned}$$

Running time of algorithm is the sum of running times for each executed statement. Above equation can be expressed in the form of $an^2 + bn + c$ for constants a, b, and c that again depends on statement costs c_i; it is thus a quadratic function of n *i.e.* n^2.

After getting the dependent tasks, scheduler checks whether this dependency is direct or indirect. For example, in figure (3.8) task T_6 directly dependent on T_7 task and T_2 is indirectly dependent on T_6 ($T_2 \rightarrow T_4 \rightarrow T_6$). These dependencies are determined by using Boolean matrix multiplication.

Input: Two copies of generating DAG of tasks for Boolean Matrix Multiplication.
Output: Dependency of tasks.

```
CHECK-INDIRECT-DEPENDENCY (matrixsize1[][],matrixsize2[][],Row, Column)

BEGIN
   1. m= ((matrixsize1.length)*(matrixsize1.length))/2
   2. for  count=0 to m
   3.   ResultMatrix=new int[matrixsize1.length][matrixsize1.length]
   4.    for  i=Row to matrixsize1.length
   5.      int [] rowVector=getCurrentRow(matrixsize1, i)
   6.     for  j=Column to matrixsize2.length
   7.         int[] columnVector=getCurrentColumn(matrixsize2, j)
   8.         for  k=0 to matrixsize2.length
   9.             if rowVector[k] == 1 && columnVector[k]==1
   10.                    ResultMatrix[i][j]=1
   11.                    flag=true
   12.                  break
   13.                End if
   14.            End for
   15.                if !flag
   16.                   ResultMatrix[i][j]=0
   17.        End for
   18.       End for
   19.          for  i=Row to matrixsize2.length
   20.             for  j=Column to matrixsize2.length
   21.             End for
   22.          End for
   23.       if ResultMatrix[Row][Column] == 1
   24.           return true
   25.         else
   26.           matrixsize1 = ResultMatrix
   27.      End for
   28.           return false
   End CHECK-INDIRECT-DEPENDENCY ()
END
```

Similar to above algorithms running time, author examine that all rows of giving matrix have $\log n$ elements, each of which is either 0 or 1. Similar examination happens with columns of the given matrix. In Boolean matrix multiplication, divide complete matrix into rows and columns and each row (column) is having $\log n$ elements. Therefore, here the complexity is $O(\frac{1}{\log n})$. The first for loop calculates the number of multiplications (number of intermediate nodes from one task to another) and within it Boolean multiplication between matrices having $O(n^2)$ complexity. Hence, the overall running time here is $O(n^3/logn)$.

This traversing of DAG gives a set of dependent and independent tasks. Further, this set adjoins the Queue of sets that works as a dispatcher. The purpose of a dispatcher is to discharge the tasks on the nodes; Task sets come in front executes in parallel on different processors and next set is dependent on that previous set. This operation dispatch sets one by one, so, it is taking $O(1)$ time.

```
Input:  Independent or dependent tasks add into a queue.
Output:  Dispatch tasks for execution.

QUEUE<SET<STRING>> QUEUEOFSET ()

BEGIN

 1. Set<String> s = Independenttaskset()
 2. if (setqueue.isEmpty())
 3.    setqueue.add(s)
 4. return setqueue

 End QUEUEOFSET ()

QUEUE<SET<STRING>> TASKEXECUTION ()

 1. Queue<Set<String>> q = queueofset()
 2. while (q.iterator().hasNext())
 3.     Taskexecution(q.element())
 4.  return setqueue
   End TASKEXECUTION ()
END
```

Above function QUEUEOFSET() add the returned set of independent task and other function dispatches the sets for execution. Tracing and dispatching of the tasks of DAG takes $O(n^3/logn)$ time in total.

3.2.4 *Assignment without Duplication*

The previous module is actual backbone of complete simulation. Dispatcher dispatches the independent tasks on the nodes and execution of the project will continue on assigned processor.

Figure (3.5) shows the clusters of processors and Table (3.1) represents the computation cost of processors with respect to tasks. Execution of tasks from the dispatcher depends on the priorities of tasks. Here, queue for a set of tasks has maintained that follows FIFO criterion.

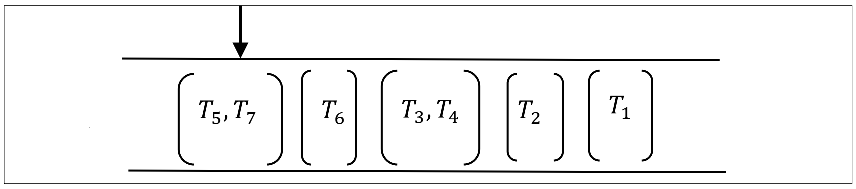


Fig. 3.9. Dispatcher Queue (FIFO)

The above dispatcher works on every processor separately. T_5, T_7 tasks will execute parallel on different processors. Now T_6 is dependent on T_7, after getting the result from T_7 , T_6 assign to other processor. T_3, T_4 are dependent on T_6. After getting output from T_6; T_3 *and* T_4 can execute in parallel. Now T_2 requires output from T_5, T_3 *and* T_4. Finally T_1executes on its own processor (source).

In figure (3.7), random tasks generate on four different processors having different efficiencies. Let us take above explained DAG that generates on high efficiency processor. In arbitrary assignment heuristic algorithms, dispatcher assigns task on other processors randomly. If its neighbor node will unable to execute more tasks than a source processor will switch to another processor.

3.2.5 *Duplication Scheduling Explanation*

This module explains the proposed duplication strategy that helps in diminution of schedule length of DAG. After the generation of DAG (task) its computational capability (efficiency) and communication cost on other processors is calculated. After bottom-up tracing of DAG dispatcher queue is maintained that initially allocate processors to the first set of independent tasks. Those assigned independent tasks can execute in parallel on allocated processors. After the execution of assigned tasks, processor of dependent tasks starts execution because output of predecessor becomes the input for its successor.

Now, for the execution of such dependents, task duplication is used. Our duplication approach is based on following factors:

1. Communication cost: Time taken in the resettlement of the predecessor output towards its successor is the communication cost between them. If this data transfer rate is high then there is a requirement of duplication.
2. Computation cost: We all are aware that the time occupied by a processor to execute the specified task is the computation cost of the assigned tasks on allotted processor.

In order to execute our approach, first we set computation costs of particular task (let us say $task_i$) on all processors in ascending order. Additionally, communication costs between $task_i$ and its successors will arrange in descending order. Afterwards, scheduler compares the successors computation cost on source processor of $task_i$ and communication cost between tasks. If computation cost is smaller than the $C_{task_i,successors}$ then duplication of successor task on source processor of $task_i$ is achievable. This way of duplication along with bottom-up approach decreases the number of duplications also. Following algorithm is explaining the conditional duplication of our approach.

```
Input: task with execution time (ET) and communication cost ($C_{t_i,t_j}$) between connected tasks.
Output: Duplicate tasks to the destination processor (DP).

BEGIN
 1. IF ($ET < C_{t_i,t_j}$)
 2.    DUPLICATE ($T_i, DP$)
 3. ELSE
 4. setqueue. TASKEXECUTION ($T_i, P$)
 END
```

During the simulation of this duplication algorithm, author suspect that the number of processors affect the schedule length of the complete DAG with or without duplication. In it, one common DAG is simulated on two different distributed systems with or without duplication. The schedule length of DAG varies from the number of processors. Author checked it for 5 and 10 processors.

Question 3.1: If we increase the number of processors in any distributed system then, can there be a need of task duplication?

Explanation: Addition of any processor in a system means accumulation of new computational power in the same. We can say that if we are increasing the number of processing powers than schedule length of DAG should become small even without duplication.

Let us assume following common DAG and two different pairs of distributed system. One system is a group of 5 processors. Other system is a group of 10 processors.

Figure (3.10) explains the computation costs of tasks on given processors of the system. This theorem explains the relation between task duplication and schedule length. In order to establish the relation between both, let us consider following two examples:

1. **Less number of processors with or without duplication:**

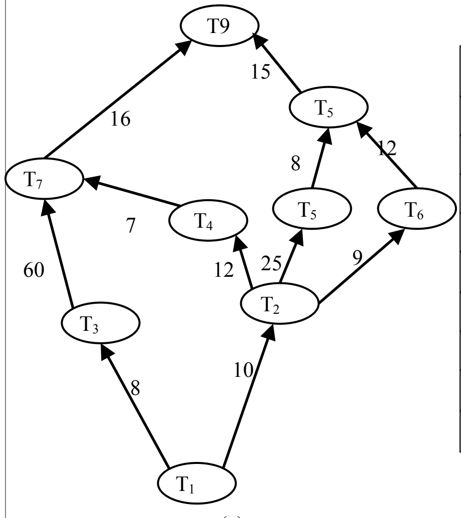
Figure (3.10(b)) is a system of 5 processors with general computational capacity. If we executes given DAG (figure (3.10 (a))) on this system by using duplication, overall schedule length of DAG is comparatively low (as shown in figure (3.11)).

2. **More number of processors with or without duplication:**

After the implementation of small system, we expand the given system by the addition of 5 supplementary processors to extra computational capacity. Following the execution of same DAG on this new arrangement, we again figure out that the schedule length of the DAG is less by using duplication.

For task duplication, author uses the following criteria:

If ($ET < C_{t_i,t_j}$) then duplication of task occurs but if reverse happens then there is no need of duplication.



(a)

P / T	P_1	P_2	P_3	P_4	P_5
T_1	25	10	6	24	12
T_2	18	15	16	20	10
T_3	9	11	8	16	14
T_4	8	9	7	10	3
T_5	12	13	15	10	8
T_6	7	13	15	12	11
T_7	19	15	20	12	10
T_8	13	20	17	14	12
T_9	16	11	15	10	12

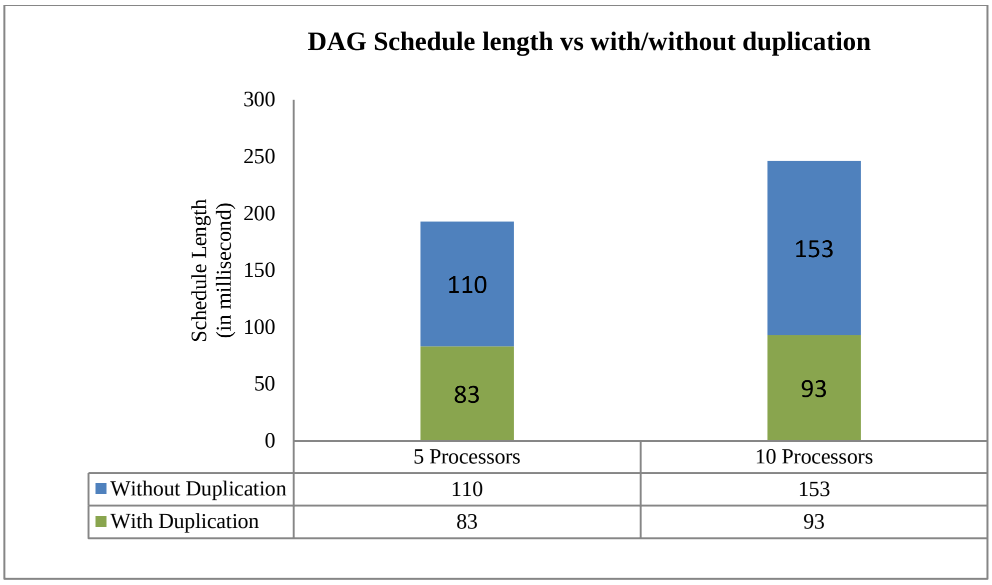
(b)

P / T	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
T_1	25	10	6	24	12	15	11	6	10	14
T_2	18	15	16	20	10	21	17	19	14	16
T_3	9	11	8	16	14	15	13	7	10	9
T_4	8	9	7	10	3	9	6	11	4	5
T_5	12	13	15	10	8	7	11	9	14	16
T_6	7	13	15	12	11	8	14	16	13	15
T_7	19	15	20	12	10	21	16	18	11	17
T_8	13	20	17	14	12	14	21	16	13	15
T_9	16	11	15	10	12	17	10	14	11	9

(c)

Fig. 3.10. (a) Arbitrary DAG (b) Distributed System of 5 (c) 10 processors

Other side of the coin is that when we increase the limit of processors by 5 then DAG schedule length has been increased as compared to 5 processor systems. Consequently, we cannot say that schedule length is dependent upon size of system. By increasing the number of processors, overall schedule length may or may not be reduced without duplication. Reason behind it, that execution of a task is dependent on computational capacity of any processor of the system and usage of duplication is best way to shorten the schedule length. Figure (3.11) is showing the result of given theorem.



	5 Processors	10 Processors
🟦 Without Duplication	110	153
🟩 With Duplication	83	93

Fig. 3.11. Schedule length vs. DAG execution with or without duplication

3.3 *Results and Comparisons*

The proposed algorithm for task duplication in heterogeneous system with mesh topology is simulated. Simulation results on Bottom-up approach of random DAG shows that the makespan generated by the proposed algorithm is better than the existing arbitrary task assignment heuristics, CAWF and HEFT-TD algorithm. The concept of Task Duplication is used in Task Assignment Heuristic in Mesh Topology. This new algorithm is named as Task Duplication Assisted Schedule Length Minimization Algorithm (TDASLM). The given example and simulations performed, explain that in the given case total finish time can be cut down by reducing the communication cost because of duplication using optimal assignment (communication cost must be greater than Execution time of related tasks on that processor).

3.3.1 *Experimental Set-up and Test Bed*

Figure (3.12) explains the experimental setup of proposed study. Following are some attributes that explain the functioning of given set-up:

1. Topology

In a distributed system, connectivity architecture follows by the processors of the entire system is known as topology. Some basic topologies followed by any network/ distributed system are BUS, Ring, Star and connected Mesh topologies. The implementation of BUS, Ring and Star topologies are simpler than connected Mesh topology. In connected Mesh topology, each processor is associated with every other processor of the system. Due to the connectivity complexity of mesh topology its handling is difficult to enforce. We simulate mesh topology in our proposed work.

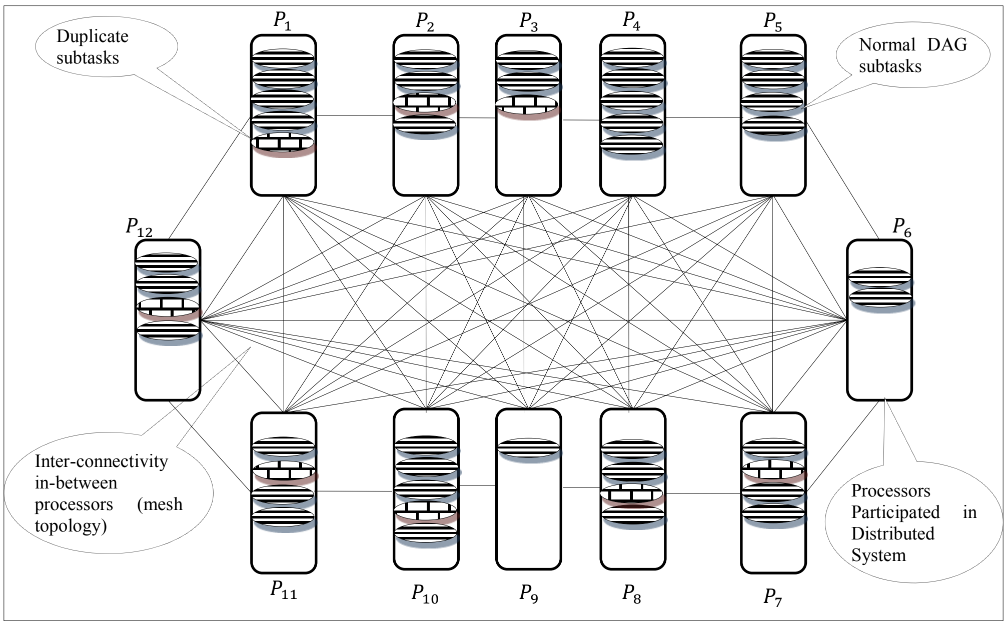


Fig. 3.12. **Experimental Set-up of Proposed Work**

2. Participating processors

Participating processors are the processor that belongs to distributed system. The participation of processors devises an environment of the system that determines the overall performance of the system. Here, heterogeneous processors are utilized in this simulation. Heterogeneous means, each processor of the system share different architecture. Internal storage capacity and computational power are the main components of any architecture. In heterogeneous, every processor has different computational capacities. Hence, we have used clustering method that split the entire system into three clusters i.e. Low, Medium and High. All clusters have some fix range of computational efficiency (Figure (3.5)).

3. Normal DAG subtasks

The proposed duplication algorithm is working on DAG. As previous section discussed that independent tasks will execute in parallel on different processors. Those assigned tasks behave like normal executable tasks.

4. Duplicate subtasks

The entire DAG (task) is divided into dependent or independent tasks. Duplication methodology is used to decrease the communication cost between dependent tasks. There are various methods for task duplication but the way of processor selection for the execution of duplicate tasks/subtasks varies. This proposed algorithm compares the computation and communication cost of duplicated task on destination processor. If its computation cost on processor is greater than the communication cost than there is no requirement of duplication.

These above techniques and all components of the framework are implemented in Netbeans 6.9 IDE environment running with Ubuntu Version 11.10. Periodic generation of random DAG takes place on any processor. Matrix is used to execute the DAG and Queue data structure has been used to implement the dispatcher. Java threads are used to execute and communicate subtasks with each other. 100 DAG's up to 30 times on 12 and 16 processors run in order to compute the overall schedule length of DAG. With this new duplication algorithm CAWF, Arbitrary task assignment heuristics and HEFT-TD scheduling algorithms are simulated on above designed framework.

Proposed algorithm is the reproduction of HEFT-TD [P. Chaudhuri, 2010], but it is implemented by using mesh topology (loosely coupled distributed system) and bottom-up approach. Therefore, its complexity is high.

Table 3.2. Algorithmic Complexity of Existing Duplication DBUS, HEFT-TD and Proposed TDASLM Algorithm

Duplication Algorithms	**Complexity**
DBUS	$O(\|n^2\|\|P^2\|)$
HEFT-TD	$O(\|V^2\|(p+d))$
TDASLM(Proposed Algorithm)	$O(\frac{n^3}{logn})$

Mesh topology is good for a limited number of processors. As processors increases connectivity's between them also increases, due to which system becomes more complex. It is the limitation of proposed algorithm that this algorithm is finer for inadequate size of distributed system.

3.3.2 *Comparisons:*

3.3.2.1 *Schedule Length:*

Schedule length (TFT) of DAG is computed by using the equation (3.3). Total finish time of DAG without duplication (Arbitrary processor selection method) is very high as compared to CAWF where schedule length is decreased by cutting down the communication cost in-between tasks. When duplication is used, the resultant schedule length is very low as compared to CAWF and arbitrary method as well. As HEFT-TD method uses top- down approach in DAG traversing in multiprocessor (tightly coupled distributed system) and proposed algorithm employs a bottom up approach on loosely coupled distributed system. Therefore, schedule length of proposed algorithm gives alike or little bit well results than other two algorithms.

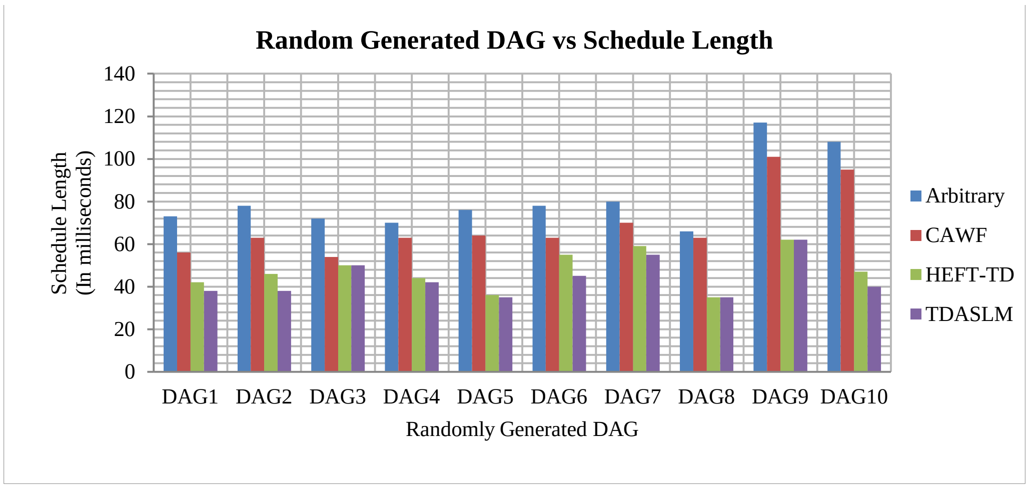


Fig. 3.13. Comparison of proposed algorithm with existing assignment algorithms

3.3.2.2 *Computation to communication Ratio (CCR):*

Computation to communication ratio (CCR) is the ratio of number of calculations a process does to the total size of the messages it sends. This ratio depends upon the average communication volume and average task execution weight. Speed of communication channel also affects the CCR and this speed depends on the computational speed of processors. In this chapter, author has used heterogeneous processors having different computational speed. It is a mesh topology also, so high processing power processor may connect with low processing power processor and vice versa is possible. Therefore, if any data moves from the higher efficiency processor to less efficient processor and speed of communication channel is very fast than CCR will be higher but if the speed of channel is high and computational cost of processor is very low than CCR will again get affected. Therefore, CCR is varying with both processor speeds as well as a communications channel because mesh topology with heterogeneous processors use here.

3.4 *Summary*

The task duplication concept has been employed here during assignment procedure (before implementation of tasks). This duplication reduces the total finish time of task. By theorem 3.1, author also explained that total finish time (schedule length) of task is utterly dependent upon the execution power of the processor and if duplication is used then it will generate good results. During simulation of task duplication author realizes that duplication of task too can overload a processor also. Hence, in order to overcome this overload problem author has used task migration methodology. How this migration technique is used with task duplication will be explained in next chapter. Therefore, further author extend this algorithm with task migration in Distributed System.

CHAPTER 4

METHOD FOR MIGRATION OF TASKS WITH DUPLICATION

Previous chapter discussed about the task duplication methodology that helps in reduction of schedule length of DAG and balance the load among processors. Author also mentioned about the overloading problem that can occur due to duplication of tasks. So, this chapter discusses about the solution of this overloading problem. Many authors [G. Couloris, 2001; A. Tanenbaum, 2002 & K. Nadiminti, 2006] describe distributed system in their own way and their overall philosophy may be described as "*A distributed system is a collection of various processors within a single system which work together for the termination of various tasks*". Task migration and duplication are techniques applied for the management of tasks.

Task migration and duplication are two independent concepts, but "*similarity between two is relocation of tasks*" and dissimilarity is "*one transfer the original tasks to single processor*" and "*second transfer the duplicate copies of particular task to various processors*". So, this chapter has been introduced with another novel algorithm which is the mishmash of task duplication and migration.

Task migration is the most widely studied method used to overcome the overloading of tasks on nodes by using various scheduling algorithms [T. T. Y. Suen, 1992]. Load between nodes is also balanced [L. M. Ni, 1985] by the task migration from heavily loaded node to lightly loaded node of the system. Similarly, duplication also uses to balance the load of the system by reducing the communication cost between the tasks and for the optimal assignment of tasks on nodes of the system [C. Wang, 2007].

Job of task assignment heuristics is to assign tasks to the processors and thus their executions proceed. If two dependent tasks are allotted to a same processor, then the communication cost between tasks will be nil but if those tasks are on separate processors then schedule length of complete problem will be large, because communication time is added along with execution time.

If overloading will occur during the assignment of tasks then migration of overloaded task (victim task) will done with the help of task migration method.

Task duplication reduces the communication cost between two dependent tasks but it can also create overloading situation. If duplicated task is reason behind the overloading than two options are present in front of scheduler i.e. either discard that task or migrate it. The removal of overloaded duplicate task will not affect much to the dependencies, because its execution is running on another processor. This chapter describes a method of migration to avoid the overloading problem.

Further sections explain the related work on task assignment, duplication and migration, new algorithm of task migration with example.

4.1 *Task Migration and Processor overloading*

Migration is an ability to shuffle a process/task from source node (executing node) to another destination node dynamically. There are following scenarios in which migration of tasks is beneficial:

1) Load Balancing (L)
2) Node Overloading(O)

Mathematically above two scenarios represent as L and O variables and both are dependent on each other i.e.

$$O = f(L) \; OR \; L = f(O) \qquad (4.1)$$

Equation (4.1) explains the dependency of overloading on load balancing between nodes of the system or if overloading occurs then there will be a need of load balancing.

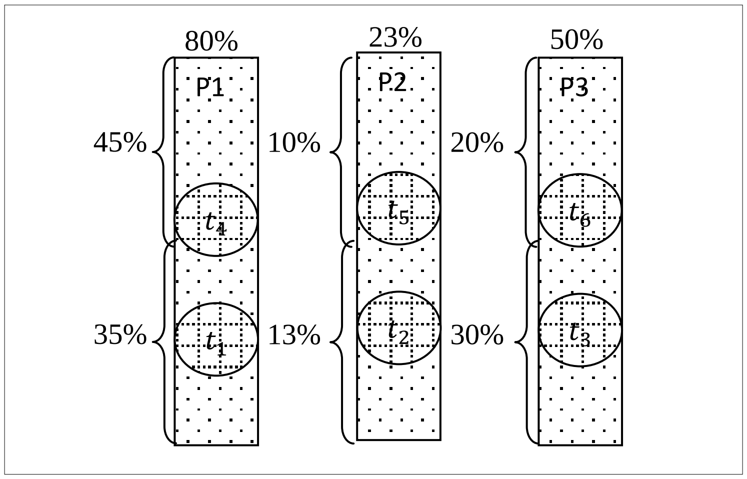


Fig.4.1. Allocation and utilization percentage of Processors

Following is the explanations of figure (4.1) with following variables and equations:

$U = 100\%$, Perfect utilization of every processor in the system.

u_i Utilization of i_{th} processor. Where $i \in 1,2,3 \ldots \ldots \ldots n$

n Total number of processors in the system.

T Total number of tasks in the given processor.

t_j j_{th} Task on particular processor.

$\{t_{1,} t_2, t_3, t_4 \ldots \ldots t_j\} \in T$

Figure (4.1) derives the following equations:

$$u_1 = u_1(t_4) + u_1(t_1) \tag{4.2}$$
$$= 45\% + 35\%$$
$$= 80\%$$

$$u_2 = u_2(t_5) + u_2(t_2) \tag{4.3}$$
$$= 10\% + 13\%$$
$$= 23\%$$

$$u_3 = u_3(t_6) + u_3(t_3) \tag{4.4}$$
$$= 20\% + 30\%$$
$$= 50\%$$

Task t_4 utilizes 45% and t_1 utilizes 35% resources of processor P1. Similarly, other tasks utilize P2 and P3 processors.

To elaborate the Load balancing scenario let us elucidate proposed algorithms:

Algorithm 4.1: The utilization factor of P1 processor is 80% and 20% are rest to meet U (100%) of P1, if any new task say t_7 will generate and it needs 25% resource utilization then it crosses 100% utilization due to which local overloading occurs.

In order to avoid overloading, task of heavily loaded node migrate to lightly loaded node (before t_7 task arrives). Now choice is to select the eligible task for migration (which should least affect system's dynamics).

Following Algorithm explains this migration as follows:

Initializations:

$P[] = \{P1, P2, P3 \dots \dots Pn\}$

$u[i] = \{u_1, u_2, u_3 \dots \dots \dots \dots . . u_n\}$

$t[j] = \{t_{1\,,}t_2\,, t_3, t_4 \dots . . t_j\}$

$CC[t_j][Pi]$ is a 2D Array or Matrix containing Computation Cost of every task on each processor. It looks like:

TABLE 4.1. Matrix of computation costs

$CC[t_j][Pi]$	P1	P2	P3
t_1	30	5	15
t_2	2	6	9
t_3	7	20	25
t_4	21	2	4
t_5	3	12	10
t_6	21	23	11

$\max_{1\le i\le n}(u_i)$: returns the heavily loaded Processor of the System.

$\min_{1\le i\le n}(u_i)$: returns the lightly loaded Processor of the System.

Above equations (4.2), (4.3) and (4.4) calculate the heavily and lightly loaded processor of the system.

$\max_{1 \le i \le 3}(u_1, u_2, u_3)$ Gives u_1 $i.e.$ 80% (Heavily loaded) *(4.5)*

$\min_{1 \le i \le 3}(u_1, u_2, u_3)$ Givees u_2 $i.e.$ 23% (Lightly loaded) *(4.6)*

Next Step is swapping of task from heavily loaded node to lightly loaded node. $i.e.$

$P[2] = u_1(t_1)$, t_1 task of P1 will assign to P2.

In this way, migration has occurred for load balancing.

Here is the completion of task migration process in 1st scenario.

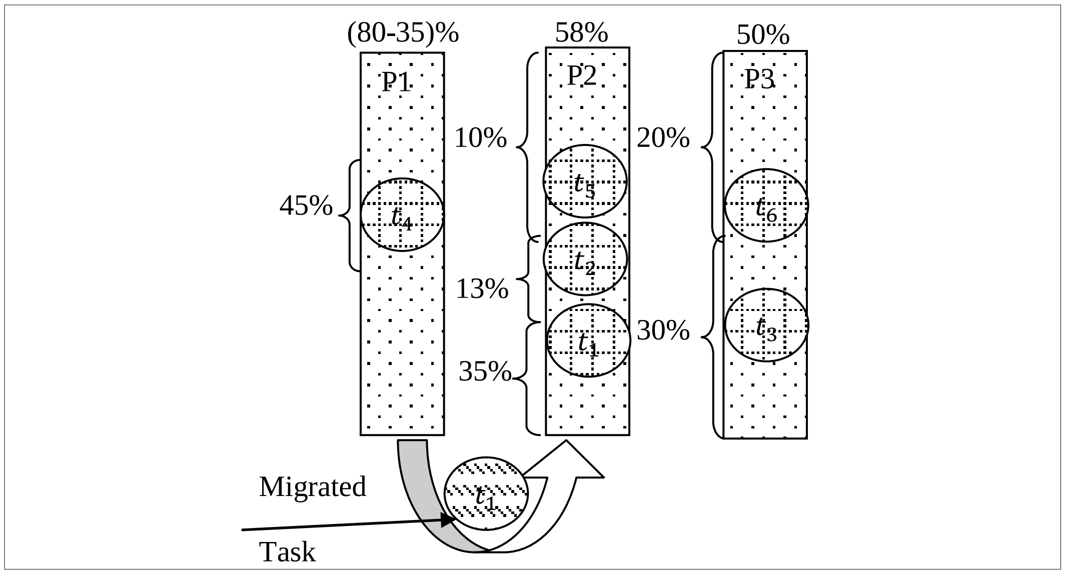


Fig.4.2. Migration of t_1 from P1 to P2

Algorithm 4.2 Overloading of task in a node.

Overloading on particular node occurs by two ways:

1) Normal Task Assignment Overloading (discussed in Algorithm 4.1).
2) Duplicated Task Assignment Overloading: If overloading on particular node will take place by task duplication then according to following conditions migration of overloaded task must occur:
    i) On basis of communication and computation cost of duplicated task on destination node.
    ii) Percentage of resource utilization on destination node.

iii) This migration will not affect the dependencies of task.

4.2 *Advantages of Migration with Duplication*

Duplication and Migration of task is used to improve the overall performance of system by load sharing. Due to load balancing, hardly system faces the overloading problem. Here a threshold limit is considered (based on Processor utilization factor, U) when crossed should invoke the Algorithm 4.1 & 4.2.

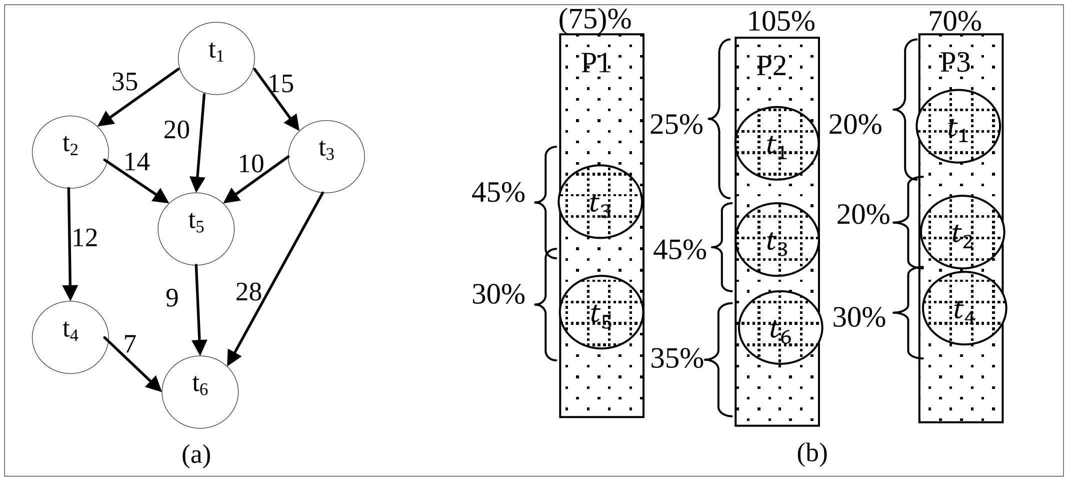


Fig.4.3. (a) Directed Acyclic Graph (DAG) (b) Task Duplication on the basis of Communication Cost

Here values of u_1, u_2, u_3 are 75%, 105% and 70% respectively. P2 processor is overloaded because it has crossed the threshold level.

Let us consider an arbitrary DAG (figure (4.3(a))) shows dependency of tasks on one another $i.e.$ $t_1 \rightarrow t_2 \rightarrow t_4 \rightarrow t_6$, $t_1 \rightarrow t_5 \rightarrow t_6$ and $t_1 \rightarrow t_3 \rightarrow t_6$; t_1 is the only predecessor of t_2, t_3 and t_5. If successor and predecessor assigned on identical processor then the communication cost between those two tasks are zero.

In given example t_1 is the predecessor of t_2, t_3 and t_5 and therefore t_1 assign on P3 and its duplicate is on P2. However, copy of task overload P2 processor. Now scheduler has two options either discard that overloaded task or migrate it to light weighted node.

Following Algorithm will follow when overloading occurs by duplicated task:

$\max_{1\le i\le 3}(u_1, u_2, u_3)$ Gives u_1 $i.e.$ 105% (Heavily loaded) *(4.7)*

$\min_{1\le i\le 3}(u_1, u_2, u_3)$ Gives u_2 $i.e.$ 70% (Lightly loaded) *(4.8)*

Step1. Compare max u_i with U, either U is less than max u_i (Overloading case) or U is greater than max u_i.

Step2. If max u_i is greater than U than calculate the overloaded percentage as follow:

Calculations of overloaded percentage *i.e.* $o\%$

$o\% = \max u_i - U$ *(4.9)*

This ($o\%$) overloaded percentage must be migrate to least weighted node.

Step3. If max u_i less than U, than before migration following selection criteria has to be used:

- *Selection of task for migration (victim task):* Task having least dependents (based on DAG) will be migrated to least weighted node.
- *Selection of destination node for migration:* For the migration of selected task, selection of destination node based on computation cost of victim task on Destination node. Computation Cost must be less than the available nodes. If migrated task is duplicate then check the communication cost between tasks of destination node with victim task.

Following example will clear the above steps of Algorithm 4.2 (if $\max u_i > U$) :

1) By using max function, selection of Source Node has done (in both algorithms).
2) Comparison between $maxu_i$ and U tells which one is greater and

$$If \quad (\max u_i > U)$$
$$then \qquad o\% = \max u_i - U$$

$o\%$ Shows the overload percentage on Source Node.

According to figure (4.3), 5% amount of load is migrated from Source (P2) to Destination (P3). $\min u_i$ shows the least weighted node in the system and this node is the Destination Node for overloaded task.

When $\max u_i < U$ in Algorithm 4.2 then victim task and Destination Node for Load Balancing has to be selected by following ways:

Following is the $function$ for the selection of victim node:

```
BEGIN
    1. $If\ (max\, u_i < U)$
    2. {
    3.     $then$
    4.         $t[j] \in max\, u_i$
    5.         $for(j = 1\ to\ n)$
    6.         {
    7.             $p\ \min_{j\le n} t[j]$
    8.             $return\ t_j$
    9.         }
    10. }
END
```

$p\ \min_{j\le n} t[j]\ function$ returns the task having least number of dependent tasks.

If that victim task is duplicated task i.e.t_3 having least dependents can be migrated on node that satisfies following attributes:

1) Lightly weighted node only.
2) Lightly weighted node with less computation cost for migrated task.
3) Least weighted node having dependent task on victim task, Hence, communication cost is another attribute.

First necessary attribute is for the selection of destination node but second and third are optional. Because it is not necessary that node having least weight contain predecessor of victim task and same case with computation cost.

In [P. Chaudhuri, 2010] method for taking the computation cost of tasks on particular nodes has been provided and accordingly node selection (based on least computation cost of victim task) is performed.

By using $\min_{1\le j\le n,1\le i\le n} CC[t_j][Pi]$ function minimum computation cost value will be retrieved and with the help of it, destination node will be assigned.
For the migration of task, selections of following three parameters are necessary:
Source node, victim task and destination node.

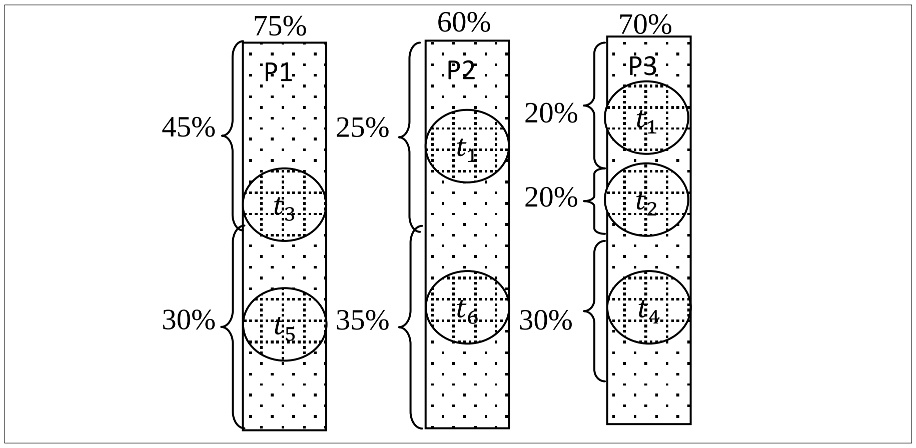


Fig.4.4. Load on processors after removal of t_3.

Example:

In figure (4.3), t_3 is a victim task that is going to migrate because this task is having only one predecessor t_6. According to the first and third attribute processor P1 is destination node.

According to algorithm 4.2 ($maxu_i > U$), 5% load of t_3 must be migrate towards processor P1 but t_3 is already there, so, discard t_3 from P2 processor. After the implementation of algorithm2 ($u_i > U$) the load on processors looks like figure (4.4).

According to algorithm4.2, where $maxu_i \le U$ and $maxu_i$ is processor P1 and here both tasks t_3 , t_5 have same number of predecessor. Ratio of load of task t_5 is less hence, victim task t_5 here. Now according to third attribute, destination node is P2 and according to second attribute destination node is P2. Therefore, t_5 task is migrating on P2 processor. After migration, u_1, u_2, u_3 become 55%, 80% and 70% respectively.

4.3 *Summary*

This chapter explains the optimal methods for the migration of duplicate task. Above example describes two algorithms for migration of duplicate and normal task. It also describes the advantage of duplication before migration in which if overloaded duplicate task is concurrently running on different processor then scheduler can easily discard that task. It also explains the important parameters for migration.

In next chapter, this task management will apply on real time tasks in which migration of tasks is dependent on the deadline of task as well as the utilization parameter. The main motive here will be the achievement of deadline without discarding of task. Therefore, Real time scheduling algorithms with their characteristics and author's proposed work will be explained in next chapter.

CHAPTER 5

REAL TIME TASK MIGRATION AND SCHEDULING ALGORITHMS

Till previous chapter, we have discussed about management of those tasks where notion of temporal correctness is weak or not significant. Now, this chapter is going to discuss on those tasks that respond to external events within a bounded interval of time that are known as real time tasks. Established on such temporal properties this real time system splits into hard and soft RTS. *Hard real time systems impose an assertion that all chores are finished within a specified time constraint. A late reply may generate a catastrophic result* [M. Joseph, 1996]. Hence, we can say that correctness in response time is a key measure of RTS. Some models of hard real time systems are nuclear power plants; embedded braking systems, avionics control systems and signal-processing systems worked for the department of defense. Supplementary, *soft real time systems has a less accurate perception of time-based correctness and the result of delayed response is not catastrophic* [M. Joseph, 1996]. Examples of soft real-time systems include electronic games, on-line transaction systems and telephone switches.

Hence, this chapter deals with real time tasks of distributed system. In chapter 1 and 2 author has discussed about scheduling algorithms of real time systems along with their advantages and disadvantages. In order to overcome disadvantage of EDF (dominos effect) and increase the success ratio, author has discussed one more algorithm that is the combination of EDF and RMS algorithm. How this algorithm works and about its architecture will be talked about in further parts of this chapter.

5.1 *An Overview of RTDS*

The system under deliberation is a Real Time Distributed System (RTDS), which is by definition "*A Distributed System having Real Time Properties*" [R. Sharma, 2013]. The architecture of processors in a distributed system can be homogenous as well as heterogeneous. In homogenous system processors share the similar architecture and dissimilar in heterogeneous. This new scheduling algorithm is implemented on homogenous system [A. Tanenbaum, 2002].

In real time system, every task has a deadline (by that time task should execute). For programming of real time tasks, RMS and EDF are two well-known scheduling algorithms under which execution of jobs based on its point of arrival or deadline as well. RMS algorithm works with static priority scheduling (offline tasks) and EDF algorithm does with dynamic priority scheduling (online tasks). The arrival of tasks in a particular system can be periodic, aperiodic and sporadic. Most systems set aside the arrival of tasks periodically because the point of task arrival is fixed and these tasks are capable to fulfill their respective deadlines (relative deadline) [J. W. S. Liu, 2000 & J. W. S. Liu, 2003].

Every scheduling algorithm has its own merits and demerits, like EDF assign priority based on deadline of the task and working well for single processor in underloading condition but working inefficiently in overloaded case [K. Kotecha, 2010]. RMS is a static algorithm and priorities are assigned on the basis of periods, but it is not as capable as dynamic algorithms for underloaded conditions [C. Wang, 2007 & K. Kotecha, 2010] but performing well in overloaded as compared to dynamic algorithms. In distributed system, this overloading and underloading problem has been balanced by using the task migration method.

For scheduling of periodic task systems on distributed system there have been two approaches: partitioning and global scheduling. In global scheduling, *single priority ordered queue of eligible tasks are maintained and scheduler selects the highest priority task for execution*. However, in partitioning scheduling, *each task is assigned to single processor deterministically and processors are scheduled independently* [J. Carpenter, 2004]. Out of these two schedulers, previously one requires task migration and in later task migration is prohibited [C. L. Liu, 1973].

Load balancing (in RTDS) is managed by using task migration with optimal scheduler and for the implementation of real time tasks either RMS or EDF algorithms are applied according to the need of the system. This chapter explains RMS and EDF together with migration.

Timely response to an event is necessary in real time system. EDF, RMS, Least laxity first, Pfair, deadline monotonic are some well-known algorithms that works well in their own perspective (discussed in former chapters). As we recognize, in EDF Domino's effect is a very usual problem that generates due to overloading condition. Similarly, RMS performs gets deprived in underloading condition [K. Kotecha, 2010 & N. W. Fisher, 2007]. We can say that both algorithms are complementary to each other. Deadline missing in both algorithms happen because of the utilization bounding approach. So, in this chapter a new scheduling algorithm will be discussed that take care of both existing algorithms drawback. Joint EDF-RM scheduling algorithm is used in the global scheduler where task migration mechanism is permissible. In order to check the superior performance of proposed algorithm, simulation on Eclipse has been carried out. Performance of the new scheduling algorithm is evaluated with few existing scheduling algorithms (EDF, RM and D_R_EDF) in terms of success ratio (SR) / failure ratio (FR), average processor utilization and maximum tardiness parameters.

5.2 *Real Time Scheduling Algorithms*

In chapter 2, author already explained basic real time scheduling algorithms (EDF and RMS). Along with these algorithms, this chapter also explains one more algorithm that is acting along the basis of EDF and RMS algorithms.

5.2.1 *D_O_EDF and D_R_EDF scheduling algorithms*

The primary motive behind the elucidation of the EDF scheduling algorithm is domino's effect problem creates due to overloading condition. We must keep in our mind that we should not let the processor shoot in such a way that causes overloading. Hence, in order to reduce this problem many authors have proposed their algorithms [J. Anderson, 2005; J. Anderson, 2008 & K. Kotecha, 2010]. D_O_EDF and D_R_EDF are one of them.

In the D_O_EDF scheduling algorithm, scheduler allocates static priorities 0 and 1 to jobs. These static priorities are further used in overloading condition. Tasks that are expected to miss the deadline scheduler discard those tasks and assign their static priority 0. Additionally, scheduler assign priority 1 to tasks having firm deadline and also expected to miss the deadline or set aside to execute [K. Kotecha, 2010] (figure (5.1)).

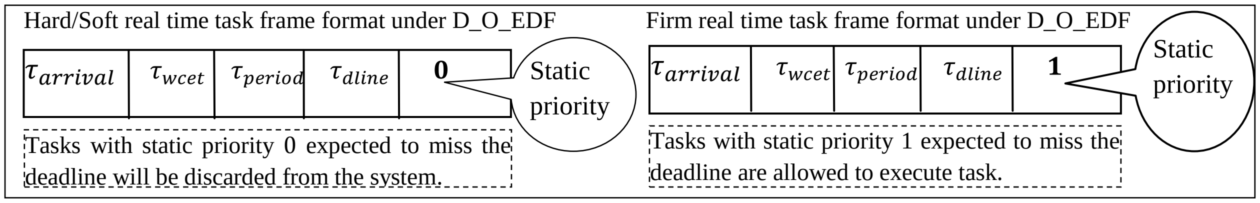

Hard/Soft real time task frame format under D_O_EDF
$\tau_{arrival}$
τ_{wcet}
τ_{period}
τ_{dline}
0
Static priority
Tasks with static priority 0 expected to miss the deadline will be discarded from the system.
Firm real time task frame format under D_O_EDF
$\tau_{arrival}$
τ_{wcet}
τ_{period}
τ_{dline}
1
Static priority
Tasks with static priority 1 expected to miss the deadline are allowed to execute task.

Fig.5.1. Real time tasks frame format according to D_O_EDF scheduling algorithm

The second algorithm D_R_EDF is a combination of dynamic and static scheduling algorithms *i.e.* EDF and RMS. [K. Kotecha, 2010] cited in his paper that EDF performs well in underloaded situation but it reduces exponentially in overloading condition. Similarly, RMS works regular in underloaded condition, but well in overloaded situation. Hence, in this algorithm primarily processor uses EDF for task performance but due to overloading as tasks start missing deadline scheduler switch towards RMS algorithm. Due to RMS when tasks continuously meet the deadline and now system is in underloaded condition then scheduler again switches towards EDF algorithm [K. Kotecha, 2010].

In the next segment, the writer will discuss about her proposed work where EDF and RM algorithm works simultaneously. Before discussing this new Joint EDF-RM Scheduling algorithm let us discuss about the architecture first.

5.2.2 *Explanation of Proposed Joint EDF-RM algorithm Architecture*

A loosely coupled distributed system is assumed here where all processors share identical architecture (homogeneous RTDS). In order to execute the tasks, EDF scheduling algorithm is employed by every processor where the threshold limit of each processor is fixed. All tasks are independent and their $\tau_{period} \geq \tau_{dline}$. Based on the priority of task preemption is allowed means higher priority job can preempt the lower priority task. Global scheduler is used in this system that maintains the global task queue for the entire system.

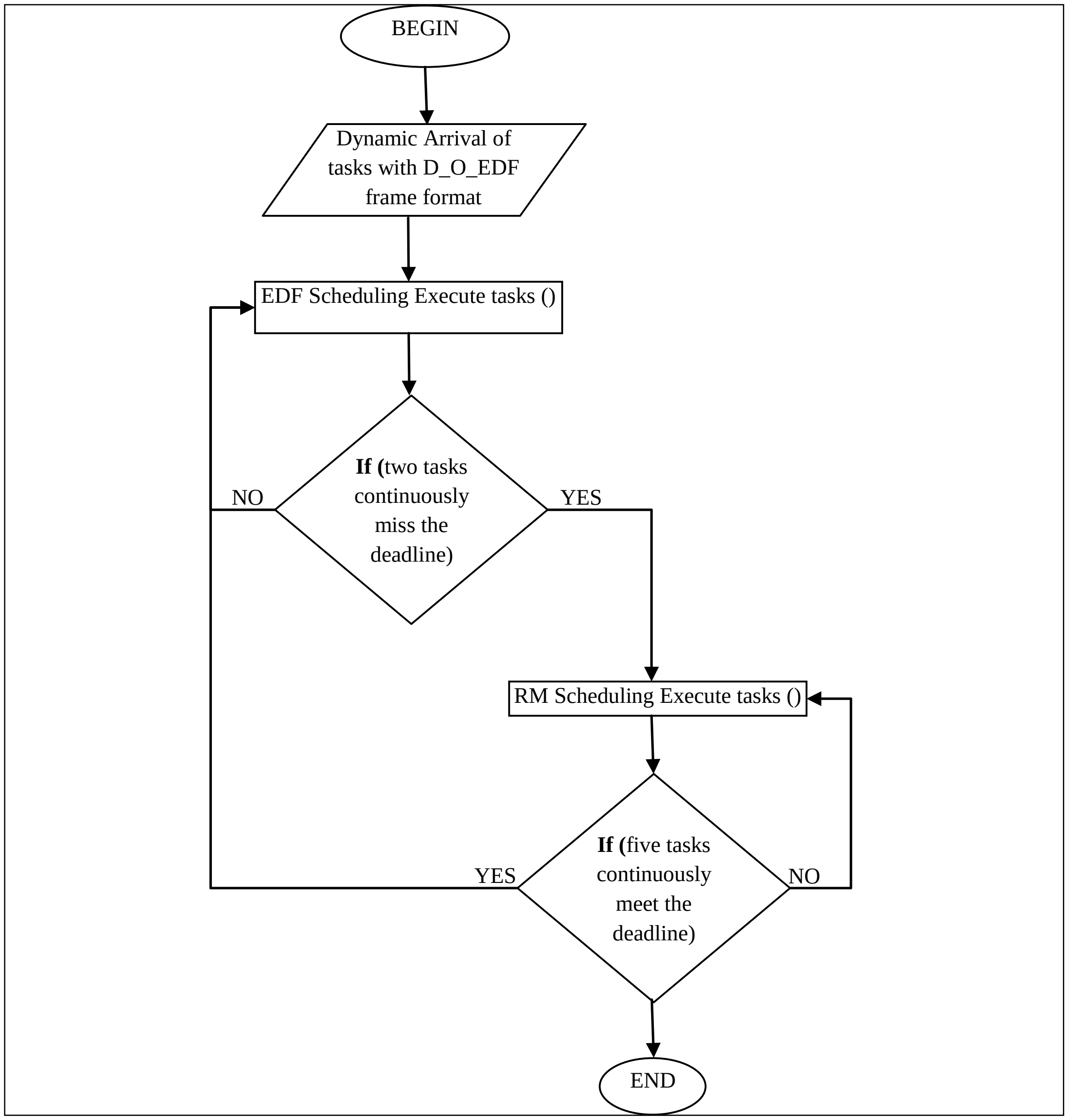

BEGIN
Dynamic Arrival of tasks with D_O_EDF frame format
EDF Scheduling Execute tasks ()
If (two tasks continuously miss the deadline)
NO
YES
RM Scheduling Execute tasks ()
If (five tasks continuously meet the deadline)
YES
NO
END

Fig.5.2. D_R_EDF scheduling algorithm Flowchart

Due to task migration permissible characteristic global scheduler is used in the proposed study. A threshold value for task migration also set in every processor based on which scheduler takes the decision whether a task is picked out for migration or not. In figure (5.3), global scheduler maintains a waiting task queue. Overloading problem is cut due to arrival of the limited amount of tasks in the queue whose boundary limit is set by using an RMS algorithm. Tasks having

$\tau_{utilization} + \sum_{i=1}^{n-1} \tau_{i\,utilization} \leq i \times \left(2^{1/i} - 1\right)$ will be easily executable or if $\tau_{utilization} + \sum_{i=1}^{n-1} \tau_{i\,utilization} \leq i \times \left(2^{1/i} - 1\right) \leq 1$ then not all tasks will be executable.

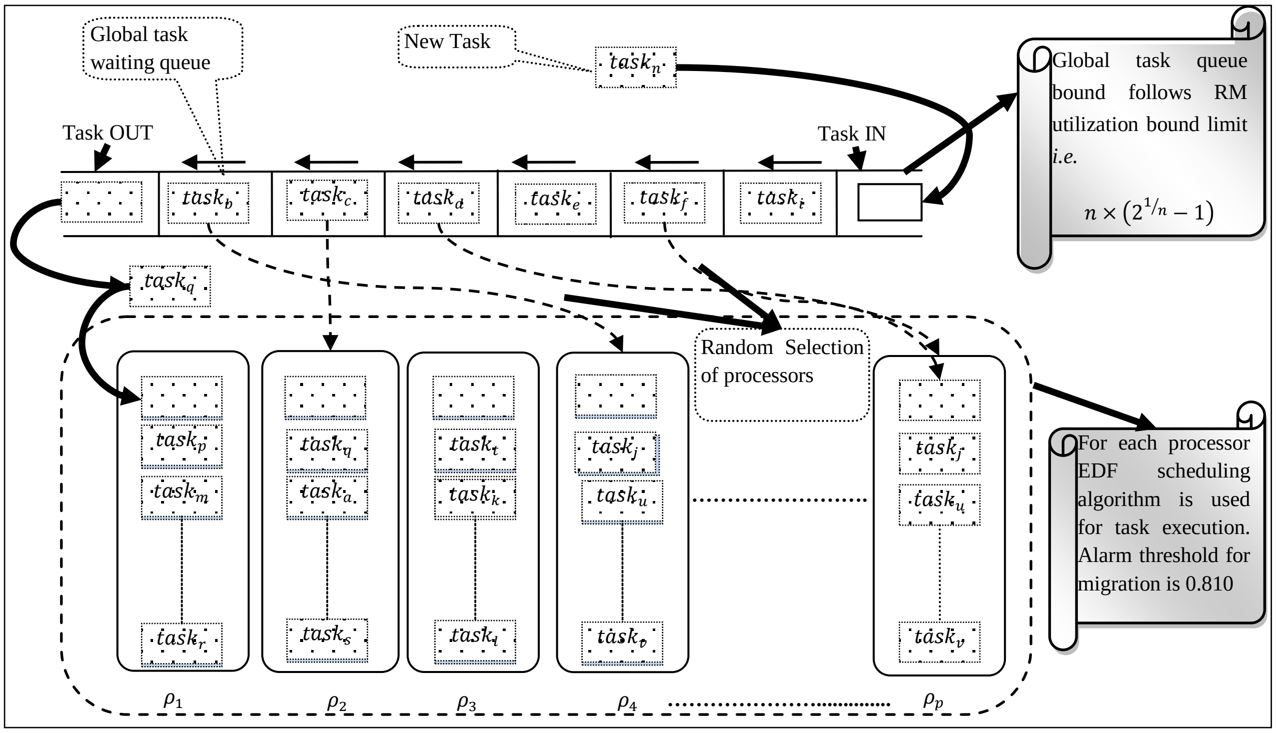
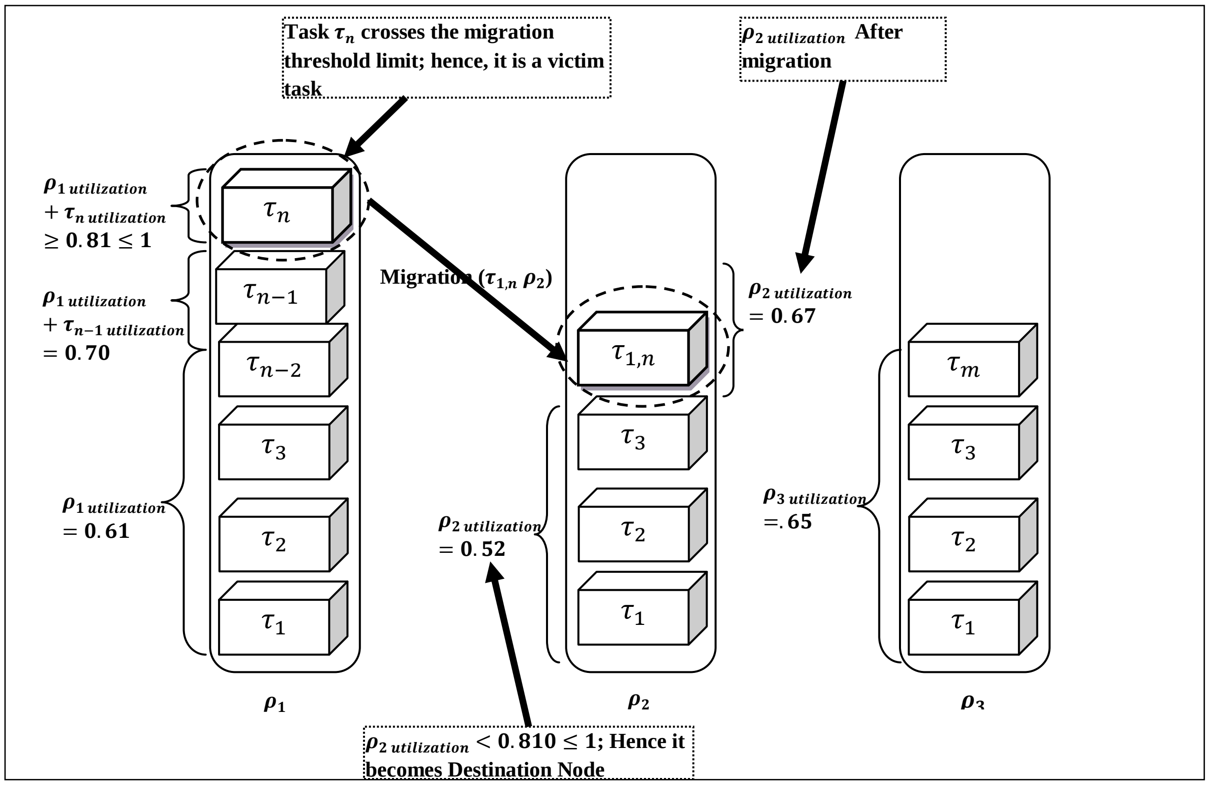

Global task waiting queue
New Task
$task_n$
Global task queue bound follows RM utilization bound limit *i.e.* $n \times (2^{1/n} - 1)$
Task OUT
Task IN
$task_b$
$task_c$
$task_d$
$task_e$
$task_f$
$task_i$
$task_q$
Random Selection of processors
$task_p$
$task_m$
$task_r$
$task_q$
$task_a$
$task_s$
$task_t$
$task_k$
$task_l$
$task_j$
$task_u$
$task_o$
$task_f$
$task_u$
$task_v$
For each processor EDF scheduling algorithm is used for task execution. Alarm threshold for migration is 0.810
ρ_1
ρ_2
ρ_3
ρ_4
ρ_p

Fig.5.3. Architecture of proposed algorithm

The working of global task queue is based on first in first out (FIFO) and tasks are randomly assigned to processors. Tasks on each processor executes with the help of EDF scheduling algorithm. In this algorithm, minute change is a migration threshold limit that determines the migration of task from is given processor. In figure (5.4), ρ_1 processor utilization is $\rho_{1\,utilization} = \tau_{i\,utilization} + \tau_{2\,utilization} + \tau_{3\,utilizaion} + \tau_{(n-2)utilization} = 0.61 \leq 0.810$ and after the arrival of τ_{n-1} processor utilization becomes $\rho_{1\,utilization} + \tau_{(n-1)utilization} = 0.70 \leq 0.810$ but arrival of τ_n reaches utilization in-between $0.810 \leq \rho_{1\,utilization} + \tau_{(n)utilization} < 1$. Continuous arriving of tasks increases $\rho_{1\,utilization}$ by 1 and afterwards all approaching tasks start missing deadline that generates domino's effect. Scheduler checks the utilization values of other processors after getting the migration threshold alarm. Later on that victim task will be migrated towards processor having least utilization. ρ_2 processor is a destination node according to figure (5.4).



Task τ_n crosses the migration threshold limit; hence, it is a victim task
$\rho_{2\,utilization}$ After migration
$\rho_{1\,utilization} + \tau_{n\,utilization} \geq 0.81 \leq 1$
$\rho_{1\,utilization} + \tau_{n-1\,utilization} = 0.70$
$\rho_{1\,utilization} = 0.61$
Migration ($\tau_{1,n}\ \rho_2$)
$\rho_{2\,utilization} = 0.67$
$\rho_{2\,utilization} = 0.52$
$\rho_{3\,utilization} = .65$
τ_n
τ_{n-1}
τ_{n-2}
τ_3
τ_2
τ_1
$\tau_{1,n}$
τ_m
ρ_1
ρ_2
ρ_3
$\rho_{2\,utilization} < 0.810 \leq 1$; Hence it becomes Destination Node

Fig.5.4. Migration Scenario in proposed algorithm

Theorem 5.1: If the upper bound of global task queue is $n \times (2^{\frac{1}{n}} - 1)$ then overloading of processor is reduced [R. Sharma, 2014].

Proof: Given set of n aperiodic tasks $\tau_1, \tau_2, \tau_3, \tau_4 \dots \dots \dots \tau_n$ arrives in a global task queue, whose periods and execution times are $\tau_{1period}, \tau_{2period}, \tau_{3period}, \tau_{4period} \dots \dots \dots \tau_{nperiod}$ and $\tau_{1wcet}, \tau_{2wcet}, \tau_{3wcet}, \tau_{4wcet} \dots \dots \dots \tau_{nwcet}$ respectively. $\tau_{1utilization}, \tau_{2utilization}, \tau_{3utilization}, \tau_{4utilization} \dots \dots \dots \tau_{nutilization}$are per task utilization. We are considering here $\tau_{deadline} \leq \tau_{period}$. There are 4 processors are present in our RTDS with $\rho_{1utilization}, \rho_{2utilization}, \rho_{3utilization}, \rho_{4utilization}$are their respective utilizations. Global scheduler randomly selects processors for the apportionment of tasks, but tasks follow FCFS discipline for allocation.

In order to proof given theorem following three cases has been hashed out:

Case I: *global queue has infinite limit*

As the global queue is containing no acceptance test of task. Without checking its utilization; based on FCFS τ_1 Task assigns to the randomly selected processor $\rho_2$whose $\rho_{2utilization} < 1$ and after the assignment of τ_1 two conditions can occur:

$$\rho_{2utilization} = \begin{Bmatrix} \rho_{2utilization} + \tau_{1utilization} & if\ \tau_{1utilization} \le 1 \\ \rho_{2utilization} & otherwise \end{Bmatrix} \tag{5.1}$$

$$\rho_{2utilization} + \tau_{1utilization} = \begin{Bmatrix} schedule\ \tau_1 & if \le 1 \\ miss\ the\ deadline\ or\ overload & otherwise \end{Bmatrix} \tag{5.2}$$

$$overload = \begin{Bmatrix} migration & if\ \rho_{nprocessor} < 1 \\ wait\ for\ execution & otherwise \end{Bmatrix} \tag{5.3}$$

In this case there are more chances of overloading on every processor.

Case II: *Boundary limit of global queue is 1 i.e.* $\sum_{i=1}^{n} \tau_{i\ utilization} \le 1$

Here only those tasks are allowed to enter in a global task queue whose $\tau_{utilization} \le 1$.

If tasks in a queue are waiting for an appointment and the arrival of new task increases the boundary limit by 1, then all upcoming tasks will not allow for entrance fee.

$$B_Q = \begin{Bmatrix} allowed\ for\ execution & if \le 1 \\ Queue\ is\ full & otherwise \end{Bmatrix} \tag{5.4}$$

In this case equation (5.1) is satisfied only 1st condition *i.e.*

$$\rho_{utilization} = \rho_{utilization} + \tau_{utilization} \tag{5.5}$$

This case gives a guarantee of schedulability of every task. Equation (5.2) and (5.3) behaves similar to 1st case. The restriction of the EDF scheduling algorithm is if one task starts missing deadline, then upcoming tasks also miss deadline continuously (Domino's Effect).

Case III: *global queue has boundary limit* $n \times (2^{\frac{1}{n}} - 1)$

Tasks having $\sum_{i=1}^{n} \tau_{n\ utilization} \le n \times (2^{\frac{1}{n}} - 1)$ are allowed to to execute on assigned processors. As we know the value of

$$n \times \left(2^{\frac{1}{n}} - 1\right) = \begin{Bmatrix} 1 & for\ n = 1 \\ < 1 & for\ n\ tends\ \infty \end{Bmatrix} \tag{5.6}$$

According to RM scheduling, every task is schedulable if its $\tau_{utilization} \le n \times \left(2^{\frac{1}{n}} - 1\right)$ but its execution is doubtful if it is in-between $n \times \left(2^{\frac{1}{n}} - 1\right)$ and 1. Therefore, here queue allows only those tasks for further execution whose $\tau_{utilization} \le n \times \left(2^{\frac{1}{n}} - 1\right)$.

$$\sum_{i=1}^{n} \tau_{i\ utilization} \le n \times \left(2^{\frac{1}{n}} - 1\right) \qquad \forall\ n \tag{5.7}$$

$$B_Q = \begin{Bmatrix} allowed\ for\ execution & if \leq n \times \left(2^{\frac{1}{n}} - 1\right) \\ tasks\ are\ non-schedulable & otherwise \end{Bmatrix} \tag{5.8}$$

After the allocation of tasks on the processor, it will execute tasks by using EDF.

$$\rho_{utilization} = \begin{Bmatrix} schedule\ tasks & if\ \leq 1 \\ overloading\ occur & if\ > 1 \end{Bmatrix} \tag{5.9}$$

However, in 3rd case very rare tasks utilization reaches towards 1, but not beyond 1. Hence, we can say that if the upper bound of global task queue is $n \times (2^{\frac{1}{n}} - 1)$ then overloading of processor gets reduced. ∎

Another reservation regarding this algorithm is the reason behind threshold limit of task migration (*i.e.* 0.81). With our experience after simulating periodic tasks, we rule that maximum number of tasks, meets their deadline when the utilization is 0.81. As in RMS, the lower bound is 0.69 and upper bound is 0.83. When we simulate EDF, we find utilization value 81% on which maximum tasks meet the deadline and that value lies between 0.69 and 0.83 *i.e.* $0.69 < 0.81 < 0.83$. Therefore, we have taken 0.81 as a threshold limit for task migration.

5.3 *Explanation of Joint EDF-RM scheduling algorithm*

In Joint EDF-RM scheduling algorithm, RMS and EDF algorithms are used in synchrony. The upper bound of processor is computed by $n(2^{\frac{1}{n}} - 1)$ in RMS where n is a number of tasks. This upper bound of RMS will set a boundary limit of global task queue B_Q of global scheduler. If collective utilization of approaching tasks is less than or equal to B_Q then tasks will distribute towards randomly selected processors of the system for execution, otherwise that task will be discarded.

An EDF scheduling algorithm is used for assigned tasks execution on a particular processor. Global scheduler helps in task migration among processors. Figure (5.4) explains the task migration methodology.

Joint EDF-RM scheduling algorithm is divided into following three modules:

1) Maintenance of global task queue
2) Execution of assigned tasks on allotted processors

3) Migration of tasks in-between processors if needed (if overloading alarm generates).

Following algorithm (5.1) explains all above-stated three modules.

Algorithm 5.1 Joint EDF-RM Scheduling Algorithm

Input: Random arrival of tasks with $\tau_{arrival}, \tau_{wcet}, \tau_{period}, \tau_{dline}$

Output: Number of tasks meet/miss the deadline along with another parameters

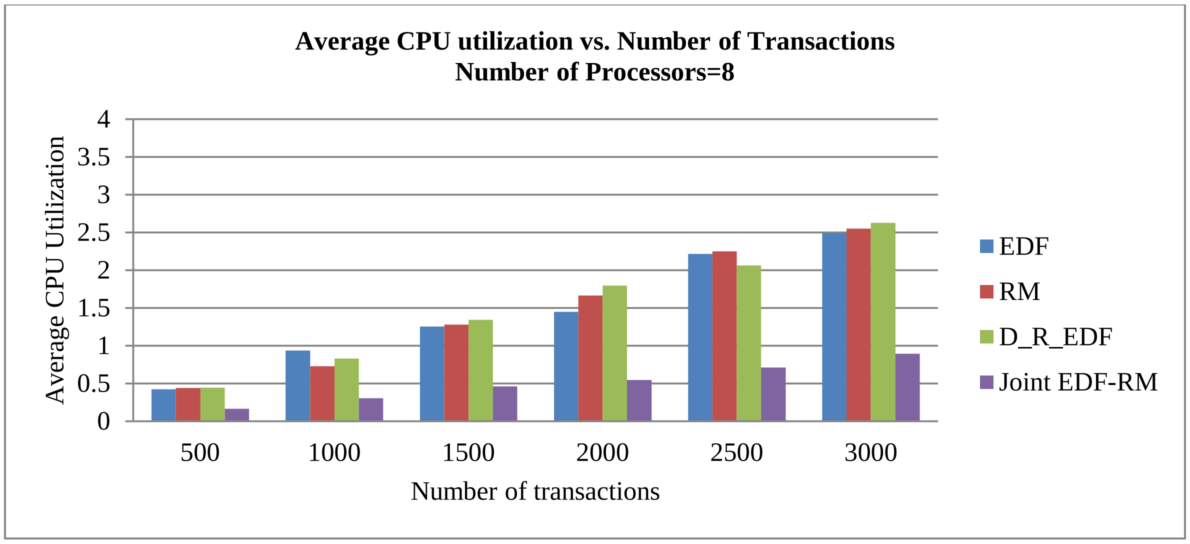
```
BEGIN

GlobalScheduler() // Global task Queue

   1. The aperiodic // Periodic arrival of tasks with arrival time, wcet,
      assigned deadline and period
   2. tasku = wcet/Period;
   3. UB=n*(Math.pow(2, 1.0/n)-1);
   4. IF tasku<=UB
   5.       Generated task is schedulable
   6.       pselection(task)
   7. Else
   8.        The task is non-schedulable

pselection(task) // Random Selection of Processors

    1. Random Selection of Processor
    2. PQueue(task);

PQueue(task) // Processors local queue

    1. Assign priorities to tasks on the basis of deadline
    2. $task_{priority} \propto \frac{1}{task_{priority}}$
    3. TaskExecution(task)

TaskExecution(task) // Task Execution by using EDF Scheduler

   1. If tasku<=1
   2.      U= U+ tasku //Cumulative accumulation of task utilization
   3. If(U<=.810)  //Processor utilization
   4.    The task is ready for execution
   5. Else
   6.  Task Migration (task, tasku)

Taskmigration(task, tasku) // Task Migration on the basis of a processor
utilization factor

    1. Sort all processor utilization
    2. Assign task to the processor having least a utilization factor
    3. PQueue(task);

    END
```

5.4 *Comparison of Joint EDF-RM with existing scheduling algorithms*

In order to evaluate the performance of given new scheduling algorithm author has used Eclipse Java EE IDE. The action of the projected study is measured by calculating the average CPU utilization, success ratio, failure ratio and maximum tardiness. Simulation is done with more than 26000 transactions in 3, 5, 8 and then 10 processors of RTDS. In simulation results, we have mentioned transactions up to 3000. Before progressing to the demonstration of calculating simulation results, let us discuss those parameters that influence the operation of joint EDF-RM scheduling algorithm with several existing algorithms (EDF, RM, D_O_EDF and D_R_EDF).

5.4.1 *Average CPU Utilization* $(A\rho_{utilization})$ is defined as

$$A_{\rho utilization} = \sum_{\rho=1}^{n} \frac{\rho_{\text{Utilization}}}{n} \qquad (5.10)$$



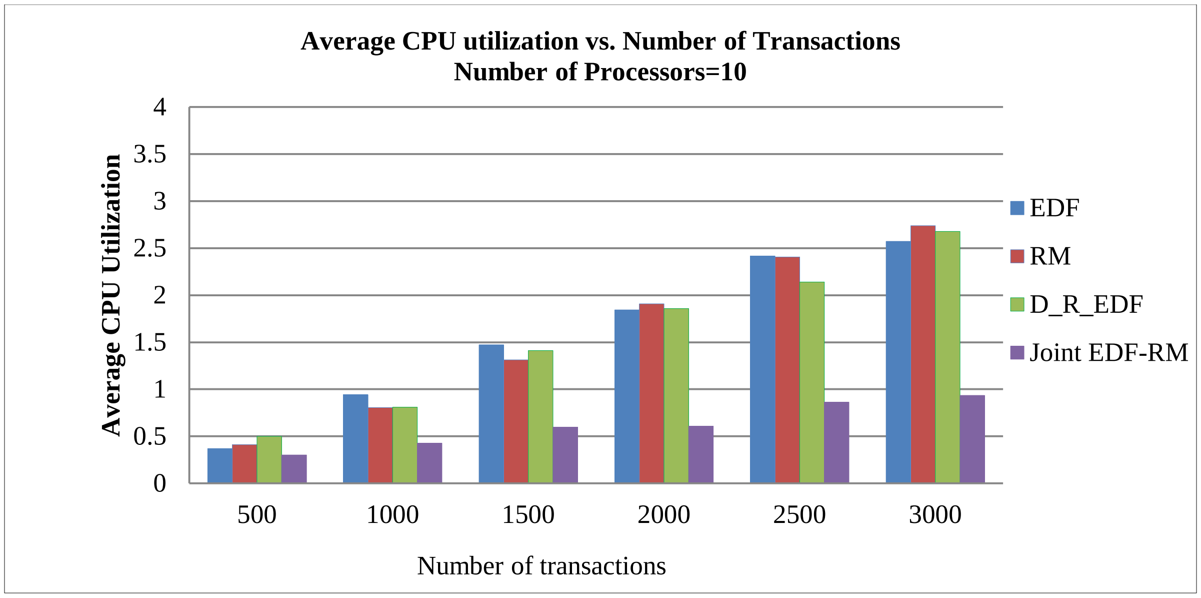


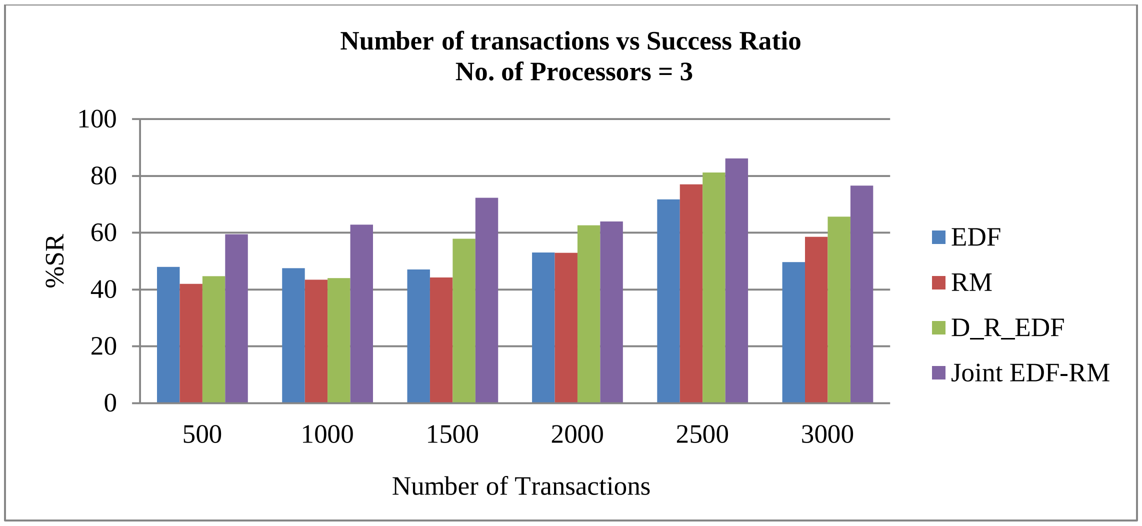
Fig.5.5. Average CPU Utilization vs. Number of transactions on 8 and 10 processors

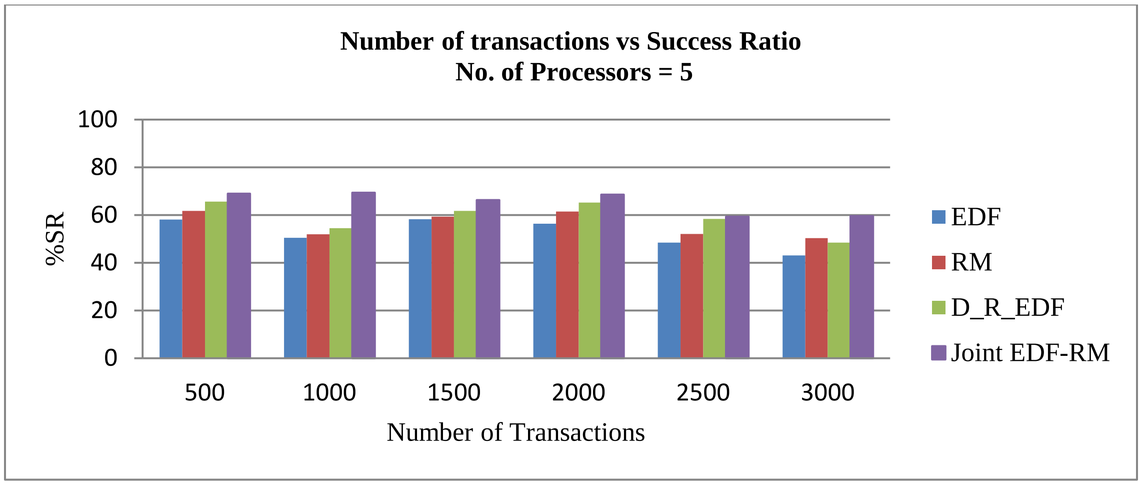
From above figure (5.5), author has simulated more than 26000 tasks on 3, 5, 8 and 10 processors. If utilization exceeds the limit by 1, it means occurrence of overloading on processors and we can observe that processor utilization in proposed Joint EDF-RM scheduling algorithm is always less than or equal to 1.

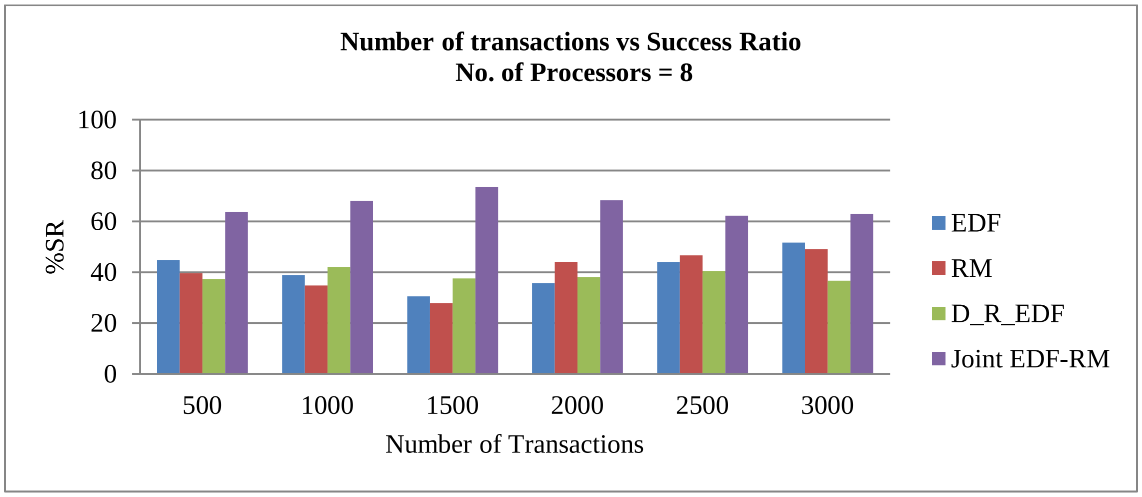
5.4.2 *Success Ratio (SR):*

$$SR = \frac{Successfully\ scheduled\ tasks}{Total\ number\ of\ tasks\ arrival} \qquad (5.11)$$

Meeting a deadline is necessary for all real time tasks, therefore author has computed success ratio that tells the percentage of successful implementation of tasks from total transactions.


Number of transactions vs Success Ratio
No. of Processors = 3
%SR
100
80
60
40
20
0
500
1000
1500
2000
2500
3000
Number of Transactions
EDF
RM
D_R_EDF
Joint EDF-RM


Number of transactions vs Success Ratio
No. of Processors = 5
%SR
100
80
60
40
20
0
500
1000
1500
2000
2500
3000
Number of Transactions
EDF
RM
D_R_EDF
Joint EDF-RM


Number of transactions vs Success Ratio
No. of Processors = 8
%SR
100
80
60
40
20
0
500
1000
1500
2000
2500
3000
Number of Transactions
EDF
RM
D_R_EDF
Joint EDF-RM

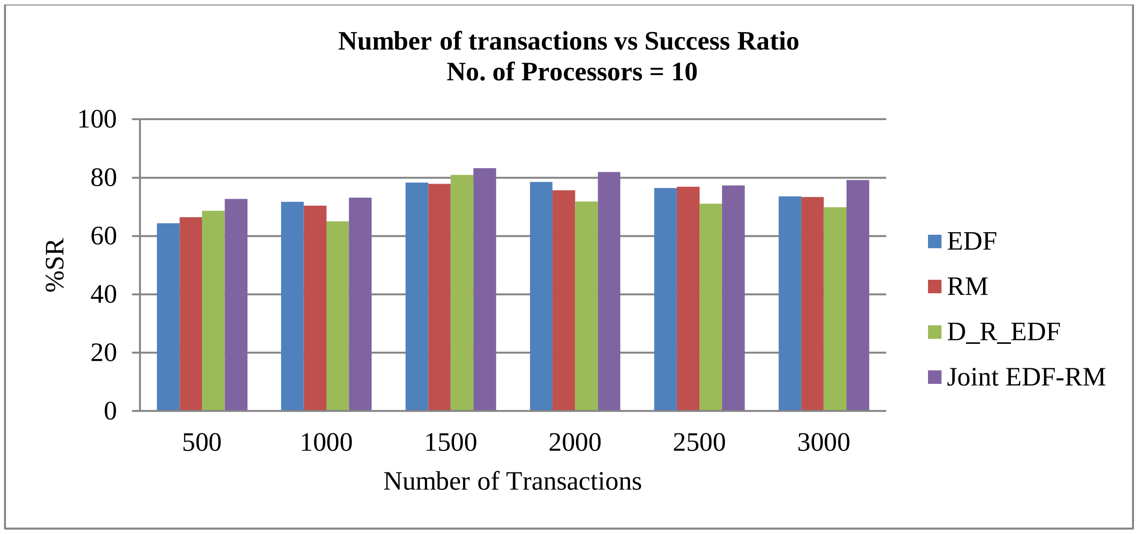


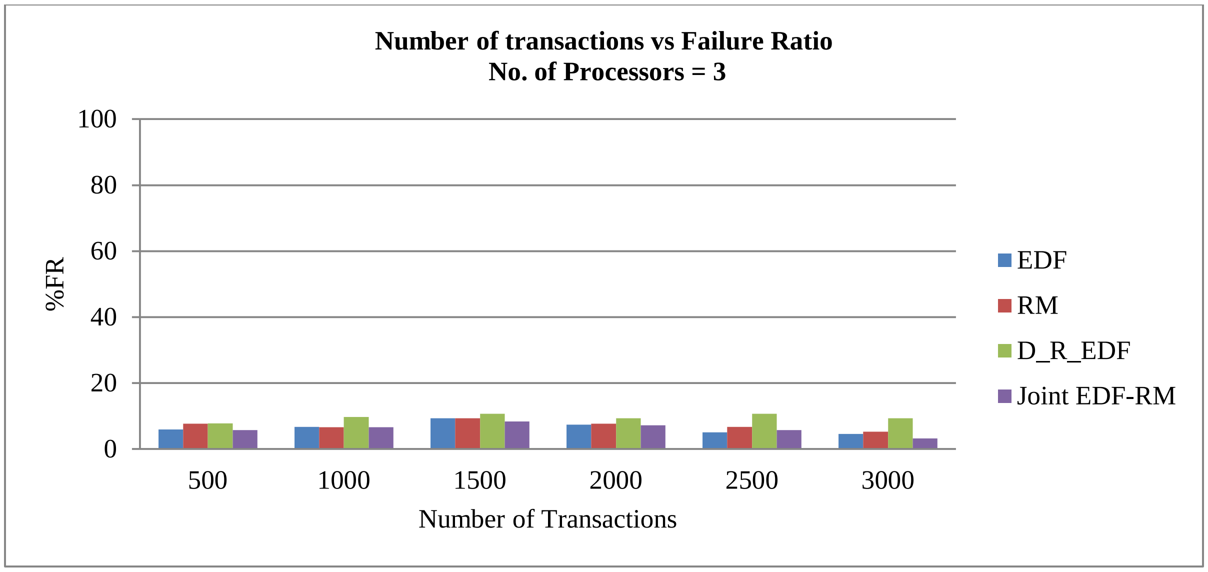
Fig.5.6. Number of transactions Vs. Success Ratio on 3,5,8 and 10 Processors

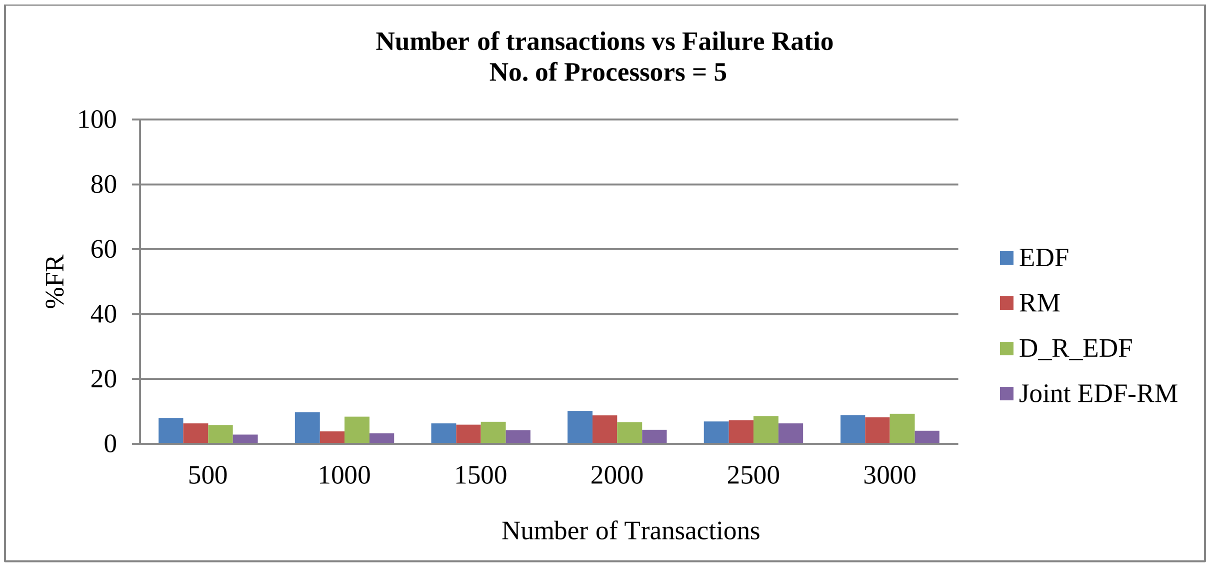
After simulating thousands of tasks author find that EDF, RM and D_R_EDF scheduling algorithm's success ratio can vary, sometimes however Joint EDF-RM algorithm has a higher success ratio. Reason behind its good performance should be credited to threshold value of task migration.

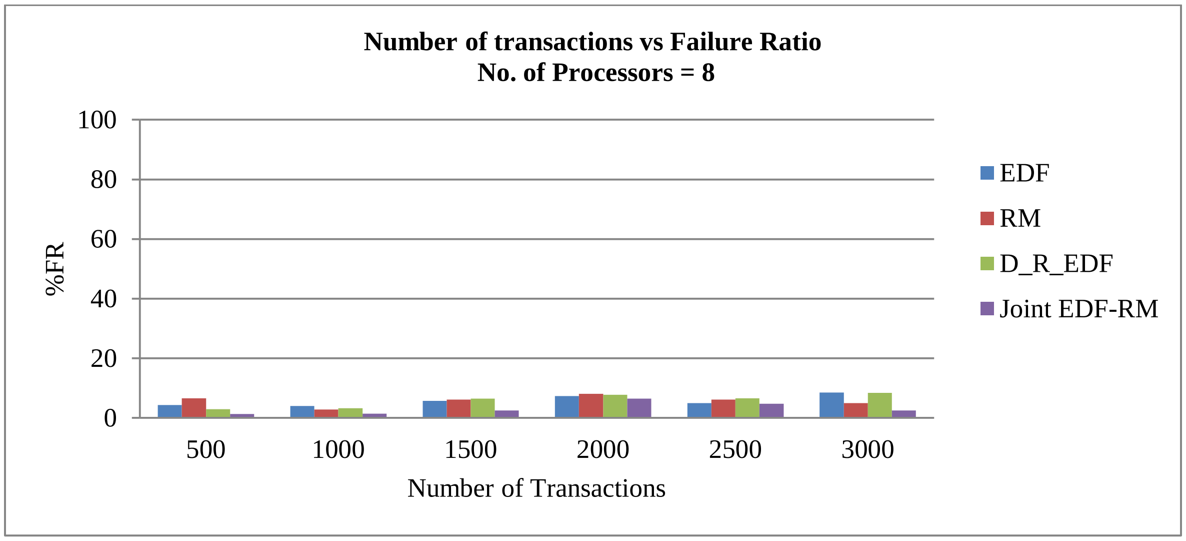
5.4.3 *Failure Ratio (FR)*

$$FR = \frac{Tasks\ Miss\ the\ deadline}{Total\ number\ of\ tasks\ arrival} \qquad (5.12)$$

This parameter computes the other phase of coin, i.e. percentage of those scheduled tasks which are unable to meet the deadline. Missing deadline is also a big task in front of all algorithms. Consequently, we also calculate the failure ratio that tells us the natural event of missing deadline.


Number of transactions vs Failure Ratio
No. of Processors = 3
%FR
100
80
60
40
20
0
500
1000
1500
2000
2500
3000
Number of Transactions
EDF
RM
D_R_EDF
Joint EDF-RM


Number of transactions vs Failure Ratio
No. of Processors = 5
%FR
100
80
60
40
20
0
500
1000
1500
2000
2500
3000
Number of Transactions
EDF
RM
D_R_EDF
Joint EDF-RM


Number of transactions vs Failure Ratio
No. of Processors = 8
%FR
100
80
60
40
20
0
500
1000
1500
2000
2500
3000
Number of Transactions
EDF
RM
D_R_EDF
Joint EDF-RM

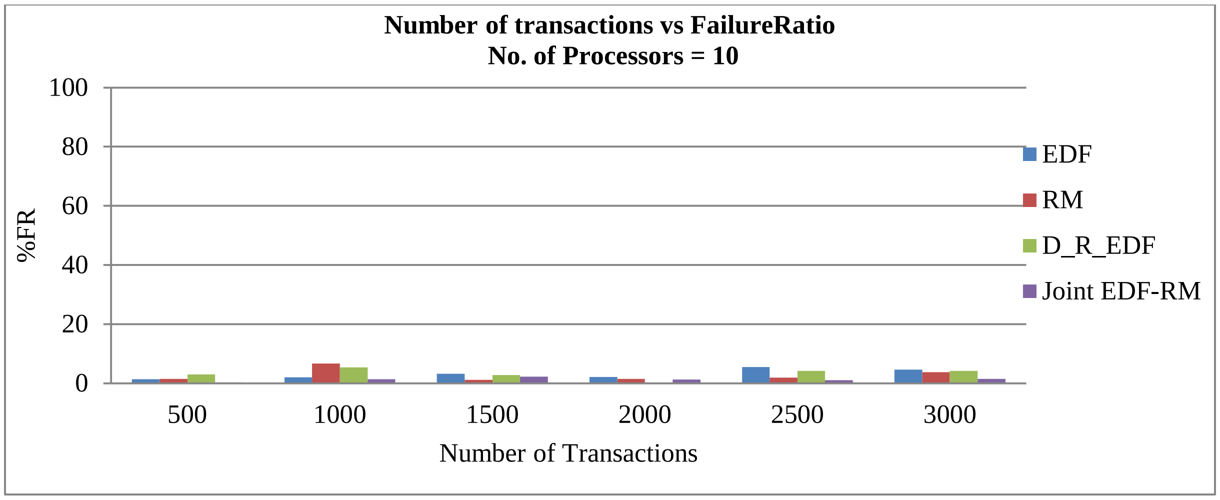


Fig.5.7. Number of transactions Vs. Failure Ratio on 3,5,8 and 10 Processors

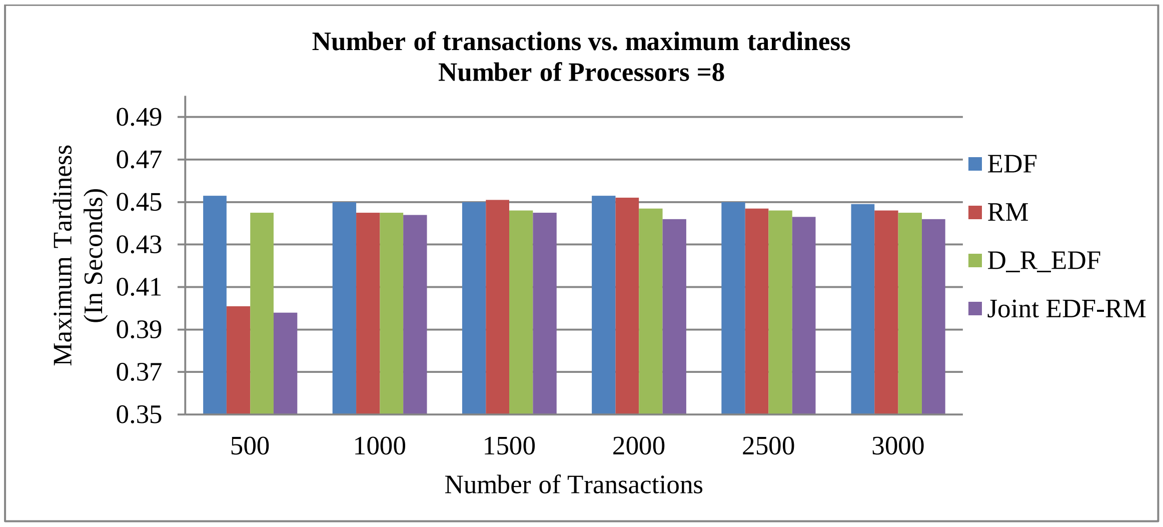
5.4.4 *Maximum Tardiness*

Tardiness is the occurrence of lateness in tasks execution, *i.e.*

$$tardiness = TST - \tau_{deadline} \qquad (5.13)$$

$$Max\ tardiness = \max(tard_{\tau_i}) \text{ where } i \in T \qquad (5.14)$$

While missing a deadline, author has computed the time after which task successfully executes on 8 and 10 processors. Figure (5.8) explains that the proposed algorithm has minimum tardiness as compared to other algorithms.



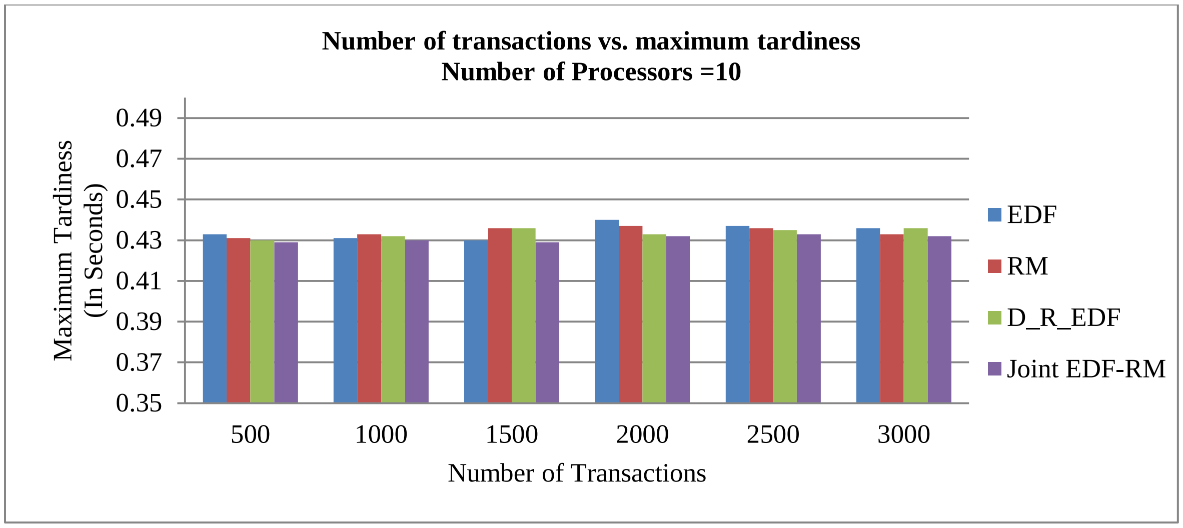


Fig.5.8. Number of Transactions vs. Maximum Tardiness on 8 and 10 processors

5.5 *Summary*

As the name of this new Joint EDF-RM scheduling algorithm explains itself that it is a hybrid of EDF and RM scheduling algorithms. It overcomes the overloading problem of any processor. Because of threshold limit 0.81 every processor generates alarm for the migration of upcoming tasks due to which overloading on task is restricted. Author simulates this work for homogeneous system; further same can be implement for heterogeneous arrangement of central processing units. One main problem happens when running tasks are preempted by higher priority new tasks due to which running tasks miss the deadline. Hence, the author also planning to work on preemption technique of programming algorithms with fault tolerant techniques. Now, the next chapter will explain how new dynamics governing parameter will replace this utilization factor.

CHAPTER 6

VISUALIZATION OF INFORMATION THEORETIC MAXIMUM ENTROPY MODEL IN RTDS

Before going through this chapter, let us take an overview on some essential key points of previous chapters. In chapter 3 and 4, we have discussed about task duplication and migration algorithms in which scheduler decides the load on given processor based on utilization parameter. On the other side, when we are dealing with real time tasks (chapter 5) scheduler checks the schedulability of given tasks on the basis of some schedulability tests and these tests also work with the help of utilization parameter. Rate Monotonic Scheduling (RMS) and Earliest Deadline First (EDF) are well-known fundamental scheduling algorithms that schedule incoming tasks with the help of processor's utilization factor. We can say that utilization factor maintains the task-load in motion hence preserves the overall working capability of processor. It is the only parameter available (been reported till date) that governs the systems dynamics of task scheduling. In this chapter author will discuss about another parameter that can take place of utilization parameter. Author has reported entropy as a candidate to govern the dynamics of RTDS task scheduling with its handling procedure.

By using real time task's information (arrival time, deadline and worst-case execution time), the probability of meeting and missing of deadline can be computed. This figured information assists to analyze entropy values of tasks/processors. Further, this entropy value can work similar to utilization with several advantages. This chapter commences another facilitator, maximum entropy model (MEM) in to action. The parameter will be desirable as it justifies the system boundary. As the confirmatory remark, the simulation results witness one to one mapping in-between both parameters (utilization and entropy). In the end, author justifies the modeled simulation with mathematical explanation of MEM in RTDS. With the help of interdisciplinary approach, we theoretically describe a new dynamics-governing stricture with some critical advantages for task scheduling of RTDS. Merit of new Entropy model over utilization factor lies in the fact that, the later considers time complexity alone. However, as in the definition of fundamental task, both time as well as space complexity has been defined. Therefore, this chapter deals with a new parameter for real time tasks scheduling algorithms. Instead of deriving new scheduling algorithms, author's main motive is to commence a new dynamics-governing factor that behaves alike utilization in existing scheduling algorithms. . Before moving towards entropy concept in depth, next section will deliver a concise preface on utilization factor and its role in RTDS followed by entropy concept.

6.1 *Utilization Factor and Entropy*

This section briefly explains the origin and role of utilization in computer science (in RTDS) followed by concept of entropy.

6.1.1 *Utilization Factor*

If we are talking in general terms that utilization is the usage or consumption of any paraphernalia for some amount of time. In language of computer discipline, we can say, "*utilization is the ratio of quantity of time to perform any task (Worst case execution time) by the maximum possible time can be used (Period)*". Real time scheduling algorithms uses utilization factor for the acceptance tests of the task schedulability. For example, in RMS scheduling, $U = n(2^{1/n} - 1) \leq 1$ is sufficient but not necessary condition [A. Mohammadi, 2005] where n is number of tasks. Similarly, in EDF $U = (\sum_{i=1}^{n} \frac{C_{task}}{P_{task}}) \leq 1$ where C_{task} is worst-case execution time (WCET) and P_{task} is Period of task. In RTS time is an important constraint by which task has to finish its execution [M. Joseph, 1996 & S. Shimokawa, 2001]. The assurance of timing activities necessitates a predictable system. Here predictability signifies that after the commencement of tasks it ought to be possible to decide the completion time of tasks with certainty. It is also pleasing that the system manage a high degree of utilization while fulfilling the timing limitations of the system [P. Penfield, 2003; G. C. Buttazzo, 2003 & J. A. Stankovic, 1988].

6.1.2 *Information Theoretic Entropy*

Following terms are required to understand and then the role of entropy in RTDS.

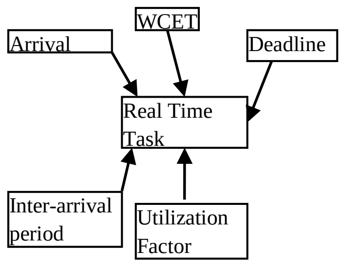
Above-mentioned paragraph notifies that timing constraint-based system needs to be predictable. Our basic aim behind the entropy concept in RTDS is predictability or presence of uncertainty in the system. Entropy is a term that is used to compute the amount of uncertainty (doubt) in the system [C. E. Shannon, 1949].

Any data that passes from source to destination for communication, encoding of any message as a sequence of independent data in cryptography, input of any random data that is used for compression algorithms in data compression or because of available information, maximum entropy principle has used for image reconstruction [S. F. Gull, 1984]. Hence, information theoretic entropy is the measure of uncertainty in random variable [D. Feldman, 2002]. In computer science, every information (data) is measured in bits, bytes etc. The arrival and execution of randomly generated tasks (information) creates uncertainty in the system. This chapter will discuss about this entropy concept and its implementation in RTDS.

6.2 *Utilization and RTDS*

So far, utilization factor is the only parameter that decides the scheduling of real time tasks in existing scheduling algorithms (RMS and EDF). All scheduling algorithms first check the acceptance test of each newly arrived task then permit particular task for the execution. We can define the utilization factor as the proportion of particular processor time spent in the execution of the task set [J. Goossens, 1999].

Following are some parameters of real time tasks:



Arrival Time	A_i
Worst Case Execution Time	$WCET_i$
Deadline	D_i
Period	P_i

After the arrival of every task, scheduler computes the utilization factor of per task: $u_i = \frac{WCET_i}{P_i}$.

Fig.6.1. Existing Attributes of Real Time Tasks

For Example: In RMS [G. Umarani, 2012] acceptance test based on utilization is $U = \sum_{i=1}^{n} u_i \leq n(2^{1/n} - 1)$ and in EDF [A. Srinivasan, 2003 & R. Sharma, 2012] acceptance test is $U = \sum_{i=1}^{n} u_i \leq 1$

Similarly, in *RTDS* these above computed utilization values decide the destination node or source node for load balancing *i.e.* task migration or duplication as well.

Liu and Layland have also defined an efficient sufficient condition for the schedulability of a task set, based on the utilization factor [C. L. Liu, 1973].

Theorem 6.1: Load Balancing in Real Time Distributed System is dependent on the utilization factor of processors of entire distributed system.

Proof: As we know that Distributed System is a system in which various processors/computers/nodes, interconnect with each other through networking topology. Suppose S is a Distributed System consists of $P_1, P_2, P_3, P_4 \ldots\ldots P_n$ and $U_1, U_2, U_3, U_4 \ldots\ldots U_n$ are their respective utilization factors.

Let us take four processors P_1, P_2, P_3, P_4 and $x_1^{'}, x_2^{'}, x_3^{'}, x_4^{'}$ are values of utilization factors of respective processors. Consider the following scenario

$$U_1 = \sum_{i=1}^{n} u_i^1 = x_1^{'} < 1 \quad (6.1)$$

$$U_2 = \sum_{i=1}^{n} u_i^2 = x_2^{'} < 1 \quad (6.2)$$

$$U_3 = \sum_{i=1}^{n} u_i^3 = x_3^{'} = 1 \quad (6.3)$$

$U_4 = \sum_{i=1}^{n} u_i^4 = x_4^{'} > 1$ *(6.4)*

Only those tasks are schedulable or accept by the scheduler whose $u_i \leq 1$

utilization of processor P_4 (equation 6.4) is greater than 1 but according to existing criteria of scheduling algorithms (EDF) utilization factor should be less than or equal to 1. Similarly, equation 6.3 explains the utilization factor of processor P_3 equals to 1 (threshold limit of U). We can say that value of U must lie between 0 to 1 *i.e.*

$0 < U < 1 \;\; where\ 0\ is\ the\ lower\ bound\ and\ 1\ is\ upper\ bound.$ *(6.5)*

Hence, if utilization factor of any processor crosses its upper bound then those tasks due to which it crosses the upper bound limit will not be schedulable. In order to balance the load; victim tasks must migrate towards those processors whose utilization value follows the equation 6.5. According to equation 6.5, if we transfer tasks from P_3 and P_4 processors towards P_1 or P_2 processor then load of entire system is balanced.

Let $y^{'}, y^{"}$ are utilization values of P_3 and P_4 processor's victim task. After the migration of these victim tasks equation 6.1, 6.2, 6.3, and 6.4 becomes:

$U_1 = \sum_{i=1}^{n} u_i^1 = 0 < x_1^{'} + y^{'} < 1$ *(6.6)*

$U_2 = \sum_{i=1}^{n} u_i^2 = 0 < x_2^{'} + y^{"} < 1$ *(6.7)*

$U_3 = \sum_{i=1}^{n} u_i^3 = 0 < x_3^{'} - y^{'} < 1$ *(6.8)*

$U_4 = \sum_{i=1}^{n} u_i^4 = 0 < x_4^{'} - y^{"} < 1$ *(6.9)*

After the migration of victim task based on utilization values the load of entire system is balanced now because utilization values of all processors lies between **0** to 1. Hence, from above derivation we can say that load balancing in RTDS is dependent on upper bound of utilization values.

The working of every real time scheduling algorithm based on utilization factor. With the help of EDF scheduling algorithm tasks schedulability are discussed on following situations:

Table 6.1. Truth table for utilization based scheduling algorithms (Uniprocessor)

u_t (per task utilization)		***U_p(processor utilization)***		***Result***
$\leq \mathbf{1}$	> 1	$\leq \mathbf{1}$	> 1	
True	*False*	*True*	*False*	*Task is schedulable*
False	*True*	*False*	*True*	*Task is non-schedulable*
True	*False*	*False*	*True*	***Task is schedulable but misses the deadline.***

In RTDS, if a task is schedulable but processor is not able to execute given task then scheduler uses migration or duplication of task for its execution on another processor. In that case, above table becomes:

Table 6.2. Truth table for utilization based scheduling algorithms (RTDS)

u_t (per task utilization)		***U_p(processor utilization)***		***Result***
$\leq \mathbf{1}$	> 1	$\leq \mathbf{1}$	> 1	
True	*False*	*True*	*False*	*Task is schedulable*
False	*True*	*False*	*True*	*Task is non-schedulable*
True	*False*	*False*	*True*	***Task is schedulable but migrates to other processor.***

As we stated before, this chapter introduces a new dynamics governing parameter *i.e.* entropy. The working of this new parameter is based on all possible information of available real time tasks or processors. Here, information of tasks are the meeting and missing deadline probability that will be computed by above stated parameters. Moreover, entire system avail the load balancing by recalculating the amount of uncertainty of all processors after the arrival or execution of any task. Yes, author is talking about the entropy concept that has been discussed until now in natural language processing (NLP), image processing, thermodynamics etc [S. F. Gull, 1984; L. Brillouin, 2004; W. H. Zurek, 1989a & W. H. Zurek, 1989b]. In further section, foundation of entropy and dynamics governing method of entire system will be discussed.

6.3 *Maximum Entropy Model and RTDS*

Up-till now utilization is the dynamics-governing factor in RTDS but here we are visualizing the entropy instead of utilization. This section explains the milieu of MEM and its groundwork in RTDS followed by utilization factor. In previous section, we have introduced entropy conceptually, now we will move towards MEM systematically.

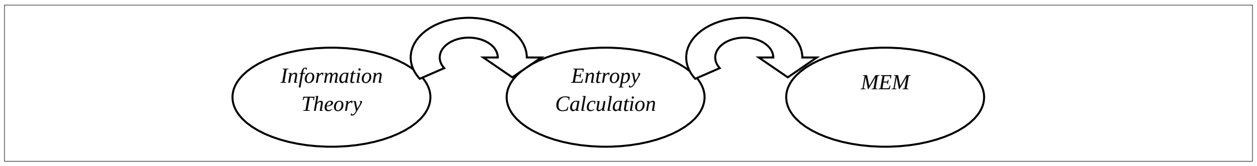


Fig.6.2. Systematic explanation of MEM

6.3.1 *Information Theory*

Shannon developed information theory in "*A mathematical theory of communication*" [C. E. Shannon, 1949]. In networking, data used for communication is information and many operations are applied on given information like compression, encryption etc. This information can be an image that moves in terms of bits 0 or 1. Since, every data in computer is measured in bits or bytes. Therefore, here the unit of information is bits or bytes. During the processing of encryption or compression of information the amount of uncertainty of given information is calculated by entropy. For example if a coin is tossed and we want to calculate the amount of information of all possible events of coin then,

$$I(head) = \log_2 \frac{1}{probability(head)} \; bits\ of\ information \tag{6.10}$$

$$I(tail) = \log_2 \frac{1}{probability(tail)} \; bits\ of\ information \tag{6.11}$$

As we are talking about distributed system, any data that moves from one system to another is information.

Statistical information theory tells the amount of information present in the event occurred. After the arrival of every task, probability of two events has occurred *i.e.* $probability(meet\ deadline)$ and $probability(miss\ deadline)$. Meeting and missing of deadline is completely dependent on time and space allocated to the task. Time is a continuous variable therefore; here we have to calculate the probabilities of events instantaneously at a unit time interval [AoPS Incorporated and Solving, 2006].

$$probability(meet\ deadline) = \frac{A_t + WCET_t}{D_t} \qquad (6.12)$$

$$probability\ (miss\ deadline) = 1 - probability(miss\ deadline) \qquad (6.13)$$

For convenience author has defined some fundamental task attributes in terms of unit space and unit time. Here for simplification author considers every task occupies unit space strictly everywhere.

6.3.2 *Entropy*

Equations (6.10) and (6.11) give the amount of information of any event and entropy will tell the amount of ambiguity being there in given information. Hence, we can define "*Entropy is the measure of uncertainty of given information*" [W. R. Derek, 2008]. This improbability is calculated by:

$$entropy(tossed\ coin) = \sum_{i=1}^{n} probability(n\ events) \times I(n\ events) \qquad (6.14)$$

Tossed coin has two events *i.e.* head or tail. Hence, (6.14) becomes

$$entropy\ (tossed\ coin) = \sum_{i=1}^{2} probability(2\ events) \times I(2\ events) \qquad (6.15)$$

$$= probability(head) \times I(head) + \ probability(tail) \times I(tail) \qquad (6.16)$$

Entropy is calculated in terms of bits. In this way, entropy incarcerates the quantity of unpredictability or improbability in any information.

6.3.3 *Maximum Entropy Model*

Information theory provides a constructive criterion for setting up probability distributions based on partial knowledge, and leads to a type of statistical inference, which is called the maximum entropy estimate. It is least biased estimate possible on the given information. It is maximally noncommittal about missing information [E. T. Jaynes, 1957; P. Penfield, 2003; Dong Yu, 2009 & R. Malouf, 2002]. Here, maximally noncommittal means that it covers all possible information diversity whatsoever. When characterizing some unknown events with a statistical model, we should always choose the probability having maximum entropy. Initially, maximum entropy concept is used in NLP area. Maximum Entropy Modeling has been successfully applied to computer vision, spatial physics, NLP and many other fields.

Again, we are taking above stated example of tossed coin. Let us consider the values of $probability(head)$ and $probability(tail)$ are .75 and .25 respectively. Then compute the entropy values with respect to each probability value

$$\begin{aligned} entropy(head) &= probability(head) \times \log_2 \frac{1}{probability(head)} \\ &= 0.75 \times \log_2 \frac{1}{0.75} = 0.75 \times 0.415 \\ &= 0.311 \end{aligned} \qquad (6.17)$$

Similarly, $entropy(tail) = 0.25 \times \log_2 \frac{1}{0.25} = 0.5$ *(6.18)*

The value of entropy in (6.18) is maximum then (6.17). $\boldsymbol{probability(tail)}$ gives the maximum entropy as compared to other one. Hence, here the maximum entropy value is $\mathbf{0.5}$. Correspondingly, for a large system we will design a maximum entropy model that decides the maximum entropy value for the entire system.

6.3.4 *Relation between Maximum entropy model and RTDS*

Information theoretic entropy is worked out with the help of information. Whatever data is processed in the system is information. In RTDS, number of arrival and execution of tasks are information for processor and all parameters of tasks are information of given task. The probability of meeting and missing of deadline of particular task is calculated by using the probability computation. These probability values tell the bits of information and from this information, per task or processor entropy will be computed. Every node has its own threshold limit of number of tasks and this threshold value decides the maximum allowed entropy of node. All nodes has its own maximum entropy value that decides the load in-between nodes. Figure (6.3) will explain the working of entire system based on entropy.

According to the principle of maximum entropy, scheduler must choose the probability that gives maximum entropy [E. T. Jaynes, 1957; P. Penfield, 2003; Dong Yu, 2009 & R. Malouf, 2002]. Every task in processors allocates some space and time for its execution. In present scenario space complexity is not a big question but time complexity (running time) of task execution matters a lot specially in RTS. Every processor has some maximum information processing capacity in per unit time. Hence, we define the maximum carrying capacity in the terms of maximum entropy of a given processor.

By using following characterizations, scheduler decides the maximum entropy of given processor:

Fundamental definition of task: An atomic task that cannot be further divided into subtasks. It occupies unit space and will take the fundamental unit time to execute.
Fundamental unit time: For a given processor or a system, the smallest amount of time span below which no information switching (task generation, task execution) can take place. For example in given system, author assumes *10-millisecond* as fundamental time unit.
Fundamental unit space: For a given processor or a system, the smallest size of allowed space below which no space allocation is possible. For example, 1 bits as a fundamental space unit.
Fundamental unit of task parameters: Arrival time(a_i), Worst-case execution time $(wcet_i)$, Deadline (d_i) and Period (p_i) are fixed parameters of any real time task. In order to compute maximum entropy author has fixed some finely granular values of given parameters. For example, each task arrives at same time with $100 - millisecond\, WCET$ and $10000 - millisecond\ deadline\ (= period)$. They are bound to follow fundamental unit space and time rule base without exception.

Based on above-stated terms and time limitations, author has done simulation by using finely granular fundamental task and computed maximum entropy values for the system. From these calculated entropy values the limit (maximum/minimum) of given processor under which working of entire system is defined. All these are theoretical explanation on entropy-based work. Further, mathematical set-up of entropy-based work has been explained.

Let us consider the hauling capability of particular processor is 100 tasks (threshold space limit). We already discussed that every task arrives at same time with worst-case execution time and deadline (already discussed). With the help of equations (6.12) and (6.13), probabilities of processor's task are:

$$probability\ (meet\ deadline) = .01\ ;\ probability\ (miss\ deadline) = \ 0.99 \qquad (6.19)$$

With these evaluated probabilities of task, entropy values for per task occurrence is calculated.

$Entropy\ (task\ meet) = 0.066$ bits and
$Entropy\ (task\ miss) = 0.014$ bits *(6.20)*

In [Dong Yu, 2009 & R. Malouf, 2002] authors has talked about how we should prefer the probability with higher entropy (or ambiguity). In above evaluated entropy values we are getting maximum entropy from $probability\ (meet)$ and therefore it will be our maximum entropy for given task.

Main motive is to decide the maximum entropy of particular processor whose maximum carrying capacity is 100 tasks. Therefore maximum entropy of given processor will be:

$$Maxentropy_p = \sum_{t=1}^{N} maxentropy_t \qquad (6.21)$$

Where N is maximum number of allowed atomic tasks (carrying capacity) on a processor, in our case it is $100\ tasks$ and therefore here the maximum entropy limit is:

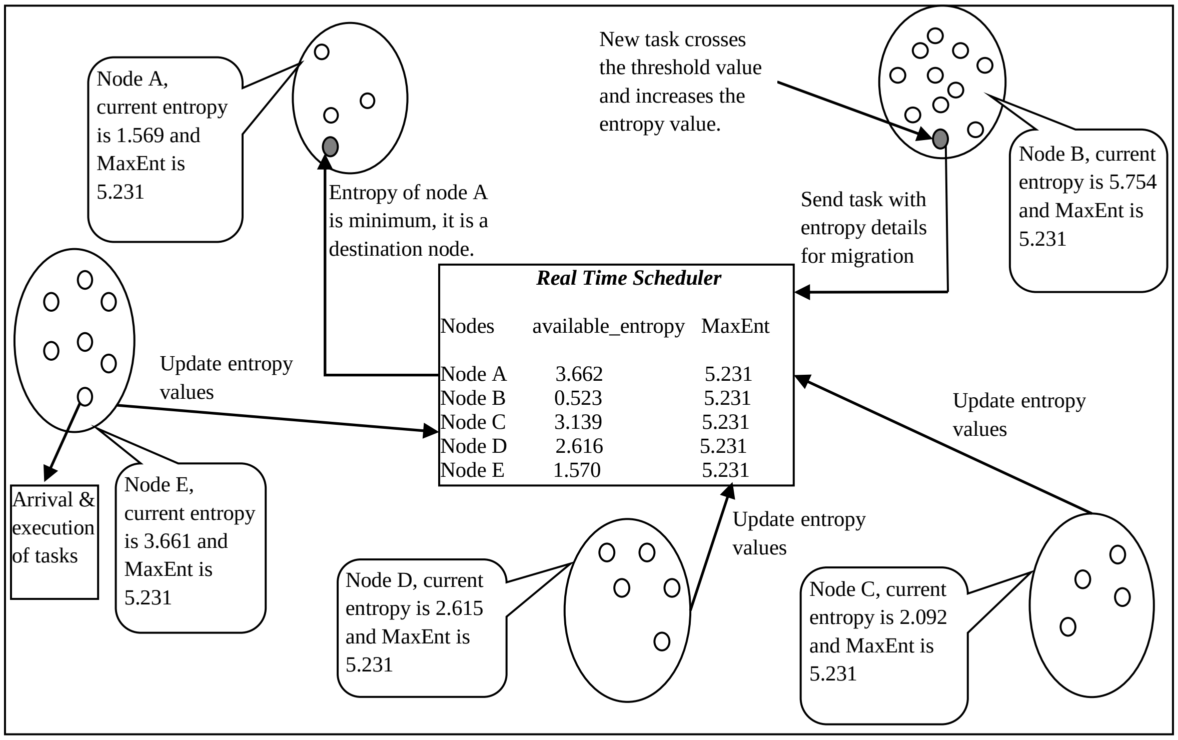
$$Maxentropy_p = \sum_{t=1}^{100} maxentropy_t \ = 6.643 \text{ bits} \qquad (6.22)$$

Here, maximum entropy is a reference standard. Now this value is used as a dynamics governing factor (in place of $U_p \leq 1$, we will take $Entropy_p \leq \text{Max}\, entropy_p$).

In this way maximum entropy model is implemented on RTDS that depends on the carrying capacity of any processor (vary from processor to processor). In a multiprocessor system, it should work in same manner as utilization factor. Further advantages of proposed maximum entropy principle over existing utilization factor are discussed.

Figure (6.3) explains the working model of RTDS based on Entropy values. There are 5 nodes present in a system. Each node has fixed threshold limit of 10 tasks. By using same fundamental concepts (already discussed), maximum entropy of each processor has been computed. Entropy is the confusion that is generated due to arrival and execution of tasks when it reaches towards its threshold limit. After the arrival and execution of task, every node updates its entropy values to the scheduler that decides the destination node at the time of migration. Node B executing 10 tasks and arrival of 11^{th} task crosses its threshold limit. When threshold limit has achieved there the higher probability of tasks missing deadline because of overloading. Therefore, when entropy value of processor crosses maximum entropy value, a selected victim task has to migrate towards another node for load balancing. In current scenario, Node B migrate chosen victim task to the scheduler and scheduler selects the node of maximum available_entropy value. Node A has maximum available_entropy value among all processors reasonably.

Difference here is that we have used maximum entropy value, instead of utilization maximum limit (that is 1). Next section will discuss the calculation of maximum entropy for every processor and the reason behind its usage in place of utilization factor.


Node A, current entropy is 1.569 and MaxEnt is 5.231
Entropy of node A is minimum, it is a destination node.
New task crosses the threshold value and increases the entropy value.
Node B, current entropy is 5.754 and MaxEnt is 5.231
Send task with entropy details for migration
Real Time Scheduler
Update entropy values
Update entropy values
Arrival & execution of tasks
Node E, current entropy is 3.661 and MaxEnt is 5.231
Update entropy values
Node D, current entropy is 2.615 and MaxEnt is 5.231
Node C, current entropy is 2.092 and MaxEnt is 5.231

Nodes	available_entropy	MaxEnt
Node A	3.662	5.231
Node B	0.523	5.231
Node C	3.139	5.231
Node D	2.616	5.231
Node E	1.570	5.231

Fig.6.3. Maximum Entropy Based Proposed model

6.4 *New Dynamics Governing Parameter*

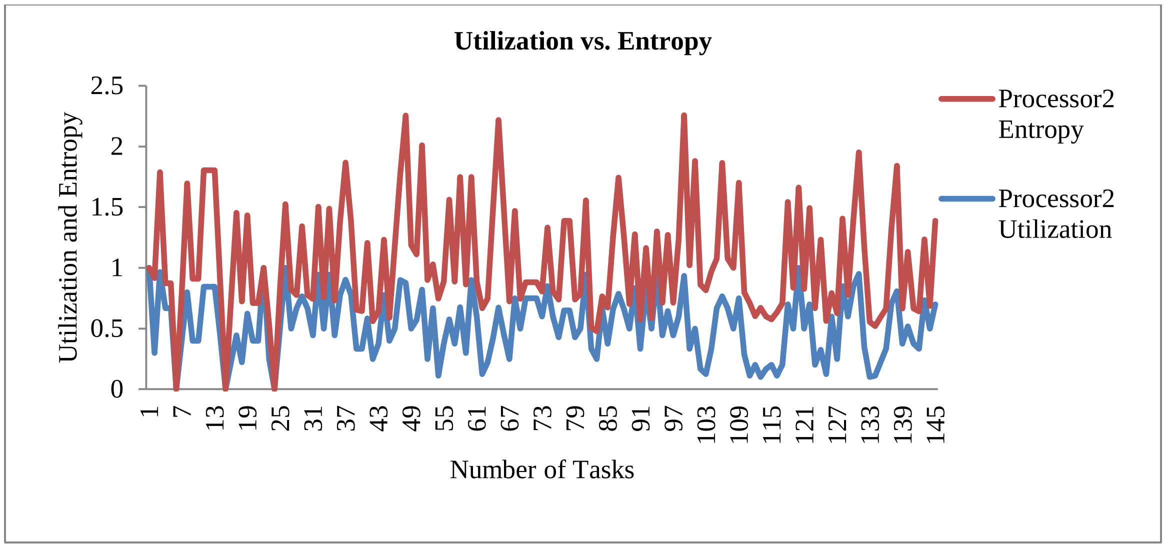
This chapter introduces new-fangled parameter entropy for the scheduling algorithms of RTDS that works according to deadline (meeting and missing) information. On the other side, in utilization factor execution of tasks depends on its resource consumption power. Consumption power here refers to the capacity of a task to occupy the processor space and time. Common property regarding both (existing and proposed) parameters is their dependency on $WCET_i, D_i, and\ P_i$. Now this section will explain the similar behavior of entropy parallel to utilization along with its advantages over utilization.

6.4.1 *One to one Mapping between utilization and entropy*:

Till date researchers derive many scheduling algorithms for real time tasks execution that works on utilization factor only [J. Singh, 2012; J. Anderson, 2005; J. Anderson, 2008; B. T. Akgün, 1996; D. R. Cheriton, 1988; M. Bertogna, 2009; A. D. Ramírez, 2012; N. W. Fisher, 2007; P. Emberson, 2007; X. Wang, 2005 & C. Lu, 2004]. Processors of the entire system achieve load balancing by checking their own utilization factor. However, now author has introduced entropy, another nominee for real time systems field. In previous sections, the working of entropy in RTS/ RTDS has been explained. We can only allow entropy in place of utilization if there is some significant similarity exist in-between both. In very simple terms if entropy behaves like utilization then only we can state that entropy is a good participant that governs dynamics similar to utilization.

As we know that maximum limit of utilization factor is 1. During simulation recalculation of processor's utilization after arrival and execution of every task, every time has been recorded. With criteria [J. Anderson, 2005] task migration using current utilization factor also handled. Similarly, by using equations (6.20), (6.21) first maximum entropy is computed and then compare current processor entropy with maximum entropy. A distributed system of 10 processors (with above said MEM criteria [Dong Yu, 2009 & P. Penfield, 2003]) is simulated for some time and take values of current entropy at several instances.

In terms of mathematics, every element of one set is allied with at least one component of another set [O'Leary, 2003] shows one to one mapping between two sets. Here, Entropy and utilization are two sets. When author simulate EDF scheduling algorithm by using entropy as well as utilization of processor parallely and plot graphs of the same then the resultant values of both follow same dynamics due to arrival and execution of tasks. Following figure (6.4) is the graph of any two processors:



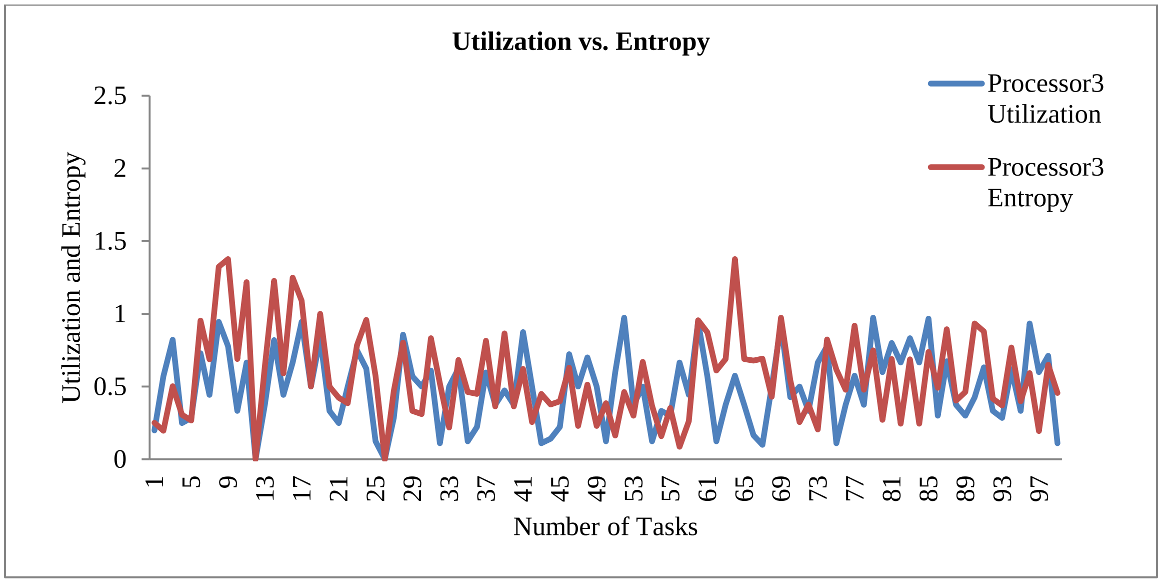
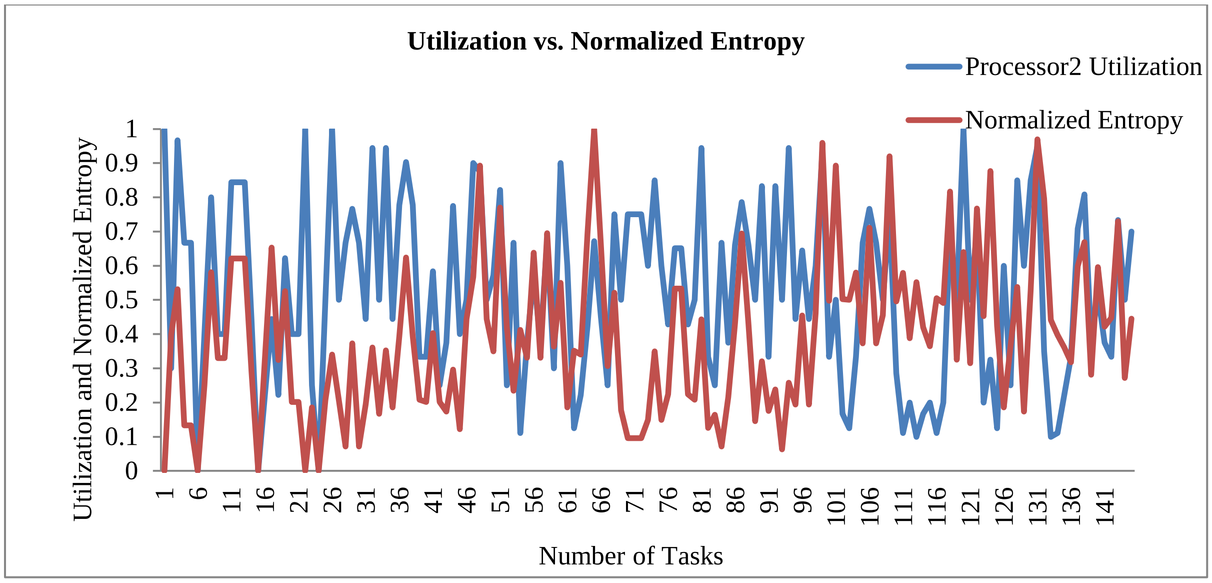


Fig.6.4. (Up and Down)Graph shows visible mapping between Entropy and utilization values

As we know that utilization is a normalized number, (values lay from 0 to 1) and entropy is a positive real number (values lay from 0 to infinity). Therefore, for affirmation, following is another plot of mapping between normalized entropy and utilization values (figure (6.5)).



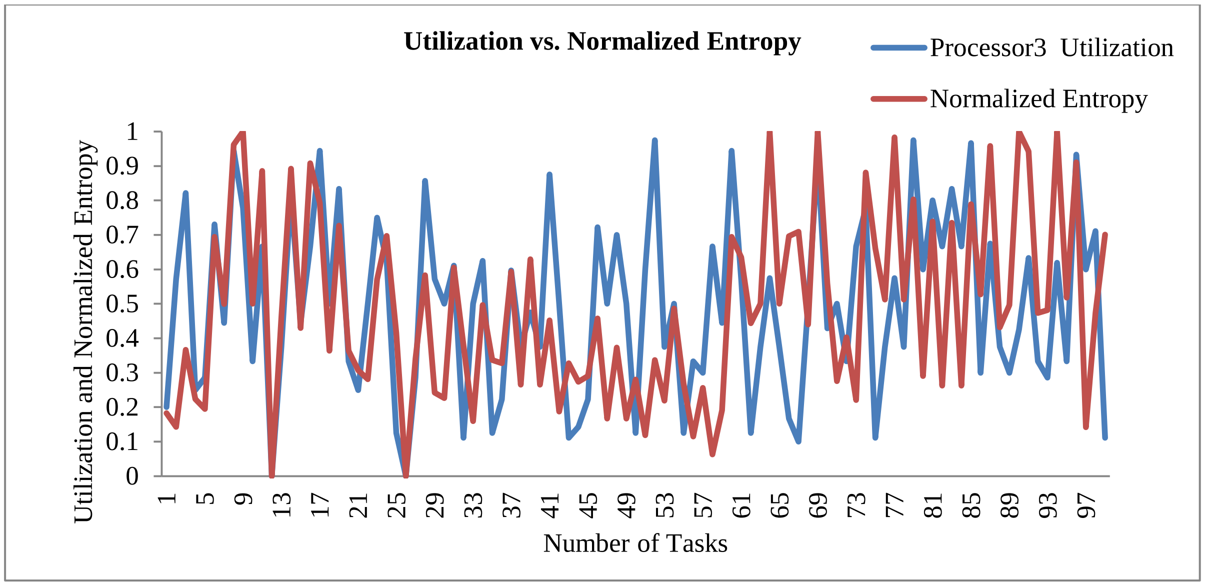


Fig.6.5. (Up and Down) Graph clearly shows one to one mapping between Normalized Entropy and utilization values

From above-stated results author want to ensure that if graph based on values of entropy and utilization follow the same pattern that generates due to arrival and termination of real time tasks then entropy can be used in place of utilization.

6.4.2 *Utilization and Entropy based Algorithm with Complexity*

As we know that utilization factor is the fraction of the amount of time used to execute the task to the maximum possible time to be used for the execution of given task. Utilization factor of per task/ processor decides the execution (schedulability) of task. Following is the algorithm for EDF scheduling:

Earliest Deadline First Scheduling Algorithm with utilization Factor (Existing)

```
---------------------------------------------------------------------------
BEGIN                                                    cost            times

SCHEDULABILITY TEST ()

    1.    if  task.Arrival+task.duration<=task.deadline  $c_1$             1
    2.        if task.utilization<=1                      $c_2$             1
    3.      U= U+ task.utilization                        $c_3$             1
    4.              if U<=1                               $c_4$             1
    5.                  EXECUTION(task)                   $c_5$             1
    6.              End if
    7.            else
    8.                TASKMIGRATION(task, task.utilization)$c_6$            1
    9.                U = U -task.utilization              $c_7$            1
    10.                     End else
    11.       End if
    12.     else
    13.   Print "task.taskname of Processor1 is Not Schedulable" $c_8$      1
    14.       End else
    15.   End if
END SCHEDULABILITY TEST ()

EXECUTION (task)

    16. while task.duration!=0                          $c_9$           $n+1$
    17.       Thread.sleep(10)                          $c_{10}$        $n$
    18.       task.Arrival=task.Arrival+10              $c_{11}$        $n$
    19.       task.duration=task.duration-10            $c_{12}$        $n$
    20. End while
    21.       stop = calendar1.get(GregorianCalendar.MILLISECOND) $c_{13}$  1
    22. if stop <=task.deadline                         $c_{14}$        1
    23.     Print  "Task meet the deadline"             $c_{15}$        1
    24.     U = U -task.utilization                     $c_{16}$        1
    25. End if
    26. else
    27.    Print "Task miss the deadline in P2"         $c_{17}$        1
    28.     U = U-task.utilization                      $c_{18}$        1
    29. End else
END EXECUTION (task)
```

```
END
```

Hence, we find that in the worst case, the running time of above scheduling algorithm is

$$T(n) = c_1.1 + c_2.1 + c_3.1 + c_4.1 + c_5.1 + c_6.1 + c_7.1 + c_8.1 + c_9.(n+1) + c_{10}.n + c_{11}.n + c_{12}.n + c_{13}.1 + c_{14}.1 + c_{15}.1 + c_{16}.1 + c_{17}.1 + c_{18}.1$$
$$= (c_9 + c_{10} + c_{11} + c_{12})n + (c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7 + c_8 + c_{13} + c_{14} + c_{15} + c_{16} + c_{17} + c_{18}) = O(n).$$

Equations (6.20) and (6.21) computes the value of $maxent$ that behaves as maximum limit for entropy value.

Earliest Deadline First Scheduling Algorithm with Entropy Factor (Proposed)

```
---------------------------------------------------------------------------
BEGIN                                                     cost           times

$maxent$ =6.64

SCHEDULABILITY TEST ()

    1.   if  task.Arrival+task.duration<=task.deadline  $c_1$              1
    2.       if task.Entropy<=$maxent$                  $c_2$              1
    3.      currentE = currentE + task.Entropy          $c_3$              1
    4.           if  totalE<=$maxent$                   $c_4$              1
    5.               EXECUTION(task)                    $c_5$              1
    6.           End if
    7.         else
    8.              TASKMIGRATION (task, task.Entropy)  $c_6$              1
    9.              currentE = currentE - task.Entropy  $c_7$              1
    10.                   End else
    11.      End if
    12.      else
    13.     Print "task.taskname of Processor1 is Not Schedulable" $c_8$   1
    14.      End else
    15.  End if
END SCHEDULABILITY TEST ()

EXECUTION (task)

    16. while task.duration!=0                       $c_9$           $n+1$
    17.       Thread.sleep(10)                        $c_{10}$        $n$
    18.       task.Arrival=task.Arrival+10            $c_{11}$        $n$
    19.       task.duration=task.duration-10          $c_{12}$        $n$
    20. End while
    21.       stop = calendar1.get(GregorianCalendar.MILLISECOND) $c_{13}$        1
    22. if stop <=task.deadline                          $c_{14}$            1
    23.     Print  "Task meet the deadline"              $c_{15}$            1
    24.     currentE = currentE - task.Entropy           $c_{16}$            1
    25. End if
    26. else
    27.    Print "Task miss the deadline"                $c_{17}$            1
    28.     currentE = currentE - task.Entropy           $c_{18}$            1
    29. End else
END EXECUTION (task)

MEMORY-PROCESSOR ()

    30. available_entropy=$maxent$-currentE              $c_{19}$            1
    [Update available_entropy of given processor to scheduler]

END MEMORY-PROCESSOR ()

END
```

Hence, we find that in the worst case, the running time of above algorithm with proposed parameter is

$$T(n) = c_1.1 + c_2.1 + c_3.1 + c_4.1 + c_5.1 + c_6.1 + c_7.1 + c_8.1 + c_9.(n+1) + c_{10}.n + c_{11}.n + c_{12}.n + c_{13}.1 + c_{14}.1 + c_{15}.1 + c_{16}.1 + c_{17}.1 + c_{18}.1 + c_{19}.1$$
$$= (c_9 + c_{10} + c_{11} + c_{12})n + (c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7 + c_8 + c_{13} + c_{14} + c_{15} + c_{16} + c_{17} + c_{18} + c_{19}) = O(n).$$

Order of complexity of algorithm with both parameters is same but entropy has following advantages over utilization:

1. *Entropy is not just a number:* As utilization is a normalized number, which just shows the system state in terms of efficiency. Besides this, no other information can be retrieved unless provided otherwise. In processors, every task requires a physical memory space in terms of bits/bytes (but not in the form of time). Entropy is measured in terms of bit/bytes (now this is space unit), while utilization is just a dimensionless quantity. Checkpoint here is the entropy can directly point out the available free space on processor.

   Refer to the code line no 30: available_entropy=$maxent$-currentE.

2. *Entropy is appreciable scaling-up factor:* Let us presume we have nodes (processors) in a complex heterogeneous clustering scenario (clustering of clusters) and we need a universal scaling up parameter that can administer the global and local (task) workload allotment. Utilization factor is doomed to fail because it is a normalized number (U<=1) and we do not use mathematical operators upon normalized numbers in a complicated scenario like this. However, there in the case of entropy we have references from thermodynamics [W. H. Zurek, 1989a & W. H. Zurek, 1989b] and information theory [S. Shimokawa, 2001] that entropy scales up easily (figure (6.6)).

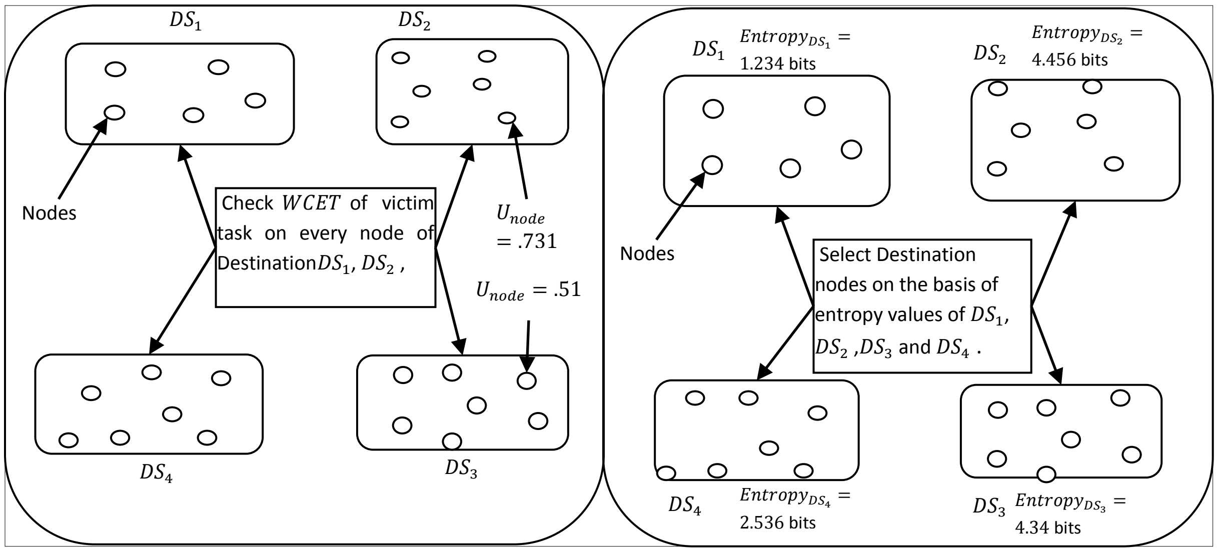

DS_1
DS_2
Nodes
Check $WCET$ of victim task on every node of DestinationDS_1, DS_2 ,
U_{node} = .731
U_{node} = .51
DS_4
DS_3
DS_1
$Entropy_{DS_1}$ = 1.234 bits
DS_2
$Entropy_{DS_2}$ = 4.456 bits
Nodes
Select Destination nodes on the basis of entropy values of DS_1, DS_2 ,DS_3 and DS_4 .
DS_4
$Entropy_{DS_4}$ = 2.536 bits
DS_3
$Entropy_{DS_3}$ = 4.34 bits

Fig.6.6. (left) Scaling with utilization (right) Scaling with Entropy Value.

6.5 *Summary*

This chapter introduces the entropy insight in RTDS field. Till date only utilization factor is used to govern the dynamics of real time tasks that works on the basis of time only [J. Singh, 2012; J. Anderson, 2005; J. Anderson, 2008; B. T. Akgün, 1996; D. R. Cheriton, 1988; M. Bertogna, 2009; A. D. Ramirez, 2012; N. W. Fisher, 2007; P. Emberson, 2007; X. Wang, 2005 & C. Lu, 2004]. Author has reported entropy as a candidate to govern the dynamics of RTDS task scheduling. Preliminary simulation study of this chapter on RTDS provides enough evidence to convince, that entropy can serve as a good competitive parameter besides conventional utilization factor. We conclude stating that entropy consumes the same order of complexity $O(n)$ when compared to utilization factor. Entropy however supersedes in the matter that its fundamental unit task definition complies with both space and time complexity at once. Furthermore, instantaneous entropy of a system or a processor carries more information (*i.e.* the free space on a given processor) that is not given by utilization factor as such. The critical advantage of this extra information provided can be used in bulk load assignment and destination processor selection. In a complex clustering scenario of multiple and heterogeneous processors this extra information would be advantageous. Next chapter will discuss on implementation of this entropy with EDF scheduling algorithm in homogeneous system with its performance. For a more complicated scenario, the same system can simulate real time dependent tasks (DAG).

CHAPTER 7

ENTROPY, A NEW DYNAMICS GOVERNING PARAMETER IN REAL TIME DISTRIBUTED SYSTEM: A SIMULATION STUDY

In previous chapter author has introduced new dynamics governing parameter that replaces utilization parameter. Along with this introduction, one to one mapping between utilization and entropy among algorithm also has been established. This chapter is going to discuss the implementation of entropy in RTDS and its performance. RTS first processes the given information then produces the outcome within a limited amount of time and if result will not generate by assigned time (deadline) then calculated tardiness either break down the system or degrade its performance. The breaking up of the system comes under hard RTS and degradation of system operation is in soft RTS [G. Umarani, 2012]. We have discussed many times that EDF and RMS [A. Srinivasan, 2003 & R. Sharma, 2012] are two very well-known and age old real time scheduling algorithms. EDF works with Dynamic tasks and RMS works with static tasks.

Author has simulated a RTDS Environment in Eclipse IDE in which periodic arrival of tasks are managed by the EDF scheduling algorithm where schedulability test is based on utilization as well as entropy. Load balancing of tasks plays a vital role in distributed systems and this important task is done by using task migration methodology. Many researchers have been working on the dynamics (task generation, execution, migration or duplication) of RTDS (or distributed system) [J. Singh, 2012; J. Anderson, 2005 & J. Anderson, 2008] and utilization is the only dynamics leading factor that is absorbed by all scheduling algorithms for system. In this chapter author replace this utilization parameter with information theoretic entropy.

7.1 *Earliest Deadline First (EDF) with utilization*

Chapter 2 discussed EDF in detail, again let us take an overview on it with the help of following example. As the name of EDF scheduling algorithm explain itself that task with least deadline is having first priority and based on their deadline tasks will placed in priority queue.

$$taskPriority \propto \frac{1}{taskDeadline} \tag{7.1}$$

Example7.1: Consider three tasks are running on a given processor with following details of Arrival Time, Execution time and Deadline.

Tasks	**Arrival time**	**Execution time (unit of time)**	**Deadline (=Period) (unit of time)**
$Task_1$	0	3	8
$Task_2$	1	2	5
$Task_3$	2	1	6

Following steps are taken while above-mentioned tasks arrive for execution on processors:

Step1. *Arrival of tasks:* The arrival of tasks follow periodic, aperiodic or sporadic patterns. For our system we consider periodic tasks in which arrival of tasks chases a fixed time pattern *i.e.* the inter-arrival time of two tasks are equal. Real time task generates with four tuples $< A_i, C_i, P_i, D_i >$ where A_i is arrival time, C_i is worst case execution time, P_i inter-arrival period and D_i is the deadline of task.

Step2. *Acceptance Test:* Before the execution of tasks, utilization factor of each task is calculated by dividing the worst case execution time (C_i) with inter-arrival period of task (P_i) *i.e.*

$$u_{t_i} = \frac{C_i}{P_i} \leq 1 \tag{7.2}$$

$$\left[u_{t_1} = \frac{3}{8} = 0.375, u_{t_2} = \frac{2}{5} = 0.4, u_{t_3} = \frac{1}{6} = .167\right] \leq 1 \tag{7.3}$$

Overall *cpu* utilization will be *i.e.*

$$U = \sum_{i=1}^{n} u_{t_i} \leq 1 \tag{7.4}$$

$$U = 0.375 + 0.4 + 0.167 = .942 \leq 1 \tag{7.5}$$

In EDF the utilization bound is of 100%. If any task does not follow equation (7.2) then that task will not be schedulable and if equation (7.2) satisfies but (7.3) not gratify then the task will not be scheduled by EDF in uniprocessor case but can be schedulable in distributed system with the help of task migration approach.

Step3. *Scheduling:* After passing the acceptance test of previous step (Step2), the task is ready for execution. During execution of tasks, if any task of higher priority arrives then already running task will be preempted by the new task of higher priority (according to equation 7.1) (figure (7.1)).

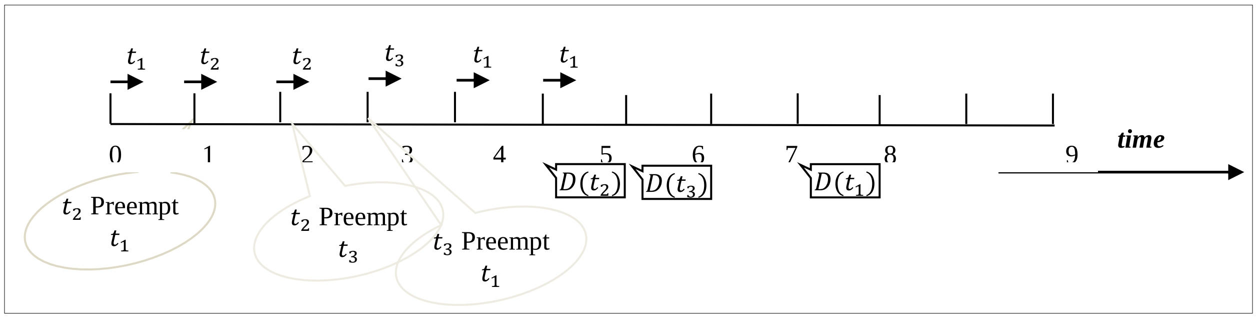


Fig.7.1. Earliest Deadline First Scheduling Algorithm

As we know that load balancing is fundamental obsession of RTDS. Load among the processors is balanced by using task migration methodology under which heavily loaded nodes transfer their load to lightly loaded nodes. The load on nodes is determined by the number of tasks running on a particular processor or by finding out its load capacity. For normal tasks of DS there is no time constraint but for real time tasks there is. Hence, in both cases utilization factor is the key that decides the overloading on the mainframe. This overloading of processor has reduced by migration of task on the processor having less utilization value. Now, during the migration of task two situations can come to play:

1. Migrated task is successfully executed on the destination processor.
2. During migration new task generates due to which processor becomes overloaded.

Due to surprisingly arrival of migrated task, processor becomes uncertain. If scheduler computes this amount of uncertainty of the processor/system, then the overall performance of the system could be increased. Consequently, this chapter establishes a concept of entropy that calculates the amount of uncertainty on a given processor. As author mentioned earlier, this chapter introduces the entropy concept as dynamics governing parameter in the domain of RTDS. Before the explanation of entropy application, we will give a few details about the entropy and information theory.

7.2. *Background of Utilization based algorithms*

This section first discusses the different ways of load balancing techniques that is explained by various researchers. After that comparison between entropy and utilization factor of given processor has been discussed. In distributed operating system, distributed scheduler works for a resource management (by load balancing) that increases the performance of the system. The implementation of load balancing algorithm is based on the measurement of processor utilization [B. T. Akgün, 1996].

In [B. T. Akgün, 2001] author explains the policies of load balancing *i.e.* transfer (host node is in appropriate state for task relocation), selection (decide tasks for transfer), location (finding a suitable transfer pair), and information (assemble load information of nodes in predestined gaps of time) policy. Author implements task migration with the help of BAG real time operating system (RTOS). The BAG RTOS is based on message passing processes like V-system [D. R. Cheriton, 1988]. In order to fulfill four basic policies of load balancing he has considered three phases: negotiation, transfer and establishment phase that is handled by three processors (host, source target module and destination target module). The selection policy of V- systems select only newly arrived tasks for transfer and its load index is the CPU-utilization at a node.

Anderson et.al [J. Anderson, 2005 & J. Anderson, 2008] proposed a new algorithm; EDF-fm based on EDF. This new algorithm places restriction on per-task utilization but no capped on overall system utilization. Author calculates the task utilization as in equation (7.2) and if $u_{t_i} \leq \frac{1}{2}$ then that task is known as light task. The execution of every project is assumed to be light because a light task can devour up to half the capability of a given processor. Under this algorithm, only enclosed number of tasks needs to be migrated and each migrated task (two subtasks of a task) will execute on two processors only. Because the victim task is divided into two subtasks and sum of the utilization factor of both subtasks equal to the total utilization of that single task. In short, this paper reduces the frequency of migration and number of migrated tasks as well. Since, the author has been working on a soft real time system; this paper also reduces the actual tardiness of tasks. The limitation of this study is that it is not being able to support dynamic task systems in which the set of tasks and their parameters preserve change at run time.

In 2009, Bertogna et. al. [M. Bertogna, 2009] designed a schedulability tests for EDF and fixed priority scheduler. Here, the authors assume that migration of task is permissible because the global scheduler is used in this algorithm. For scalability tests, the scheduler has to set some upper bound on the number of feasible task sets total utilization in a range of 4% around the point of resultant curve.

Arnoldo et. al. introduces a RealtssMP a tool [A. D. Ramírez, 2012] to perform scheduling analysis and simulation of multiprocessor real time scheduling algorithms. This tool also checks the schedulability test by calculating the utilization factor of tasks and based on these calculated utilization values migrations of tasks taken place. Similarly, in [J. Singh, 2012; N. W. Fisher, 2007; P. Emberson, 2007; X. Wang, 2005 & C. Lu, 2004] authors use utilization factors of the scheduling as well as for the migration of tasks. Author's aim behind discussing above-cited papers is to notify that up-till now every author uses utilization factor only for migration of tasks.

Moral of the story till now is that the dynamics governing parameter for task migration is utilization factor and this is a generalization. Here, author recommends switching this dynamic governing parameter to the information theoretic entropy instead of utilization factor. This chapter elucidates how this could be better in the terms of selection policy, efficiency, performance of the system etc. Entropy is the quantitative evaluation of disarray in a system. Thermodynamics is the primary causal agent of entropy that occurs due to the transfer of heat energy surrounded by a system. Similarly, in information theory term entropy is the presence of uncertainty (or improbability) in a given amount of data [W. R. Derek, 2008]. In computer science, information is a communicated data that transfer from one network (system) to another. Moreover, in cryptography entropy is the measure of uncertainty that comes after receiving of data. With the help of calculated entropy value, user can predict the presence of error in given information-set. The unit of entropy is in bits because in computers the computation on data is in the form of 0 or 1 bits [D. Feldman, 2002]. Similarly, maximum entropy calculates the amount of uncertainty in image after or before its processing [S. F. Gull, 1984].

This chapter computes the entropy value imposed on per task and processor after the arrival of tasks on given processor. Additionally, maximum entropy will control the dynamism of entire system. Immediately the question arises on entropy that how it is applicable in RTDS and is there any need of using such parameter. First, we explain the method of entropy's application and at the end of this chapter reason behind using this new parameter instead of utilization will be excused.

7.3 *Information Theoretic Entropy based Algorithm*

Entropy is the uncertainty that means indecisive point for a particular system. It creates criticality due to which no one can predict what will happen. Real time tasks generate periodically and each task executes on the basis of its priority. Tasks due to which overloading occurs are migrated towards other processors. Arrival, migration and execution of tasks create a critical condition due to following reason:

1. If newly generate task follows the schedulability test (equation 7.2) condition but there is no assurance of its successful execution on a particular processor. Might be another task of high priority either migrate or generate on that processor and preempt it. Due to which from time to time processor becomes overloaded or underloaded. The present scenario is very uncertain.
2. The execution of a task is about to finish but by mean time a new task preempts it. This situation is also very uncertain.

Hence, if the scheduler is able to compute the presence of uncertainty of system and task then above-mentioned problems can easily reduce. This chapter sheds light on the entropy concept in RTDS. In order to simulate task migration methodology by using entropy values author has simulated following three errands:

- Existing EDF scheduling algorithm in RTDS.
- Entropy in place of utilization can be used if and only if there should be some similarity between both. Therefore, in previous chapter author has shown one-to-one mapping in-between both entities.
- Implementation of the EDF scheduling algorithm by using per task or processor entropy values in RTDS.

7.4 *Utilization Based Task Migration*

7.4.1 *Mathematical Explanation of EDF scheduling algorithm in Distributed System scenario*

Let us assume DS is a distributed system having j number of processors. $DS = \{\rho_1, \rho_2, \rho_3, \rho_4 \ldots\ldots \rho_j\}$ also written as

$$\rho_j \in DS \tag{7.6}$$

T is a task set of n number of independent tasks $T_i = \{t_1, t_2, t_3, t_4, t_5 \ldots\ldots\ldots . t_n\}$ that arrives on a particular j^{th} processor ρ_j, we can say that:

$$T_i \in \rho_j \tag{7.7}$$

Now calculate overall utilization of processor $\rho_j(U)$ that cannot be computed without calculating the per task utilization factor which must be less than or equal to 1, $u_{t_i} = \frac{C_i}{P_i} \leq 1$

$$u_{t_i} \quad \begin{Bmatrix} if\ \leq 1 & task\ is\ schedulable \\ Otherwise & discard\ the\ task \end{Bmatrix} \tag{7.8}$$

$$U = \sum_{i=1}^{n} u_{t_i} = u_{t_1} + u_{t_2} + u_{t_3} + u_{t_4} \ldots\ldots\ldots u_{t_n} \tag{7.9}$$

$$U \quad \begin{Bmatrix} if \leq 1 & tasks\ will\ be\ executable \\ Otherwise & tasks\ unable\ to\ meet\ their\ respective\ deadline \end{Bmatrix} \tag{7.10}$$

In the similar way, overall utilization of other processors of the system will be computed. Equation (7.8)-(7.10) calculates the utilization factor of per task and per processor as well. Further equations explain the load balancing methodology:

Let us take the computed value of u_{t_i} *is* x and it lies between $0 < x \leq 1$. X is the value of U that also lies between $0 < X \leq 1$. Let us assume following scenario:

After the arrival of task t_i value of U_{ρ_j} is X $(X \leq 1)$ and utilization factor of next task t_{i+1} is

$$u_{t_{i+1}} = x' \quad and \quad x' \leq 1 \qquad (7.11)$$

But value of $u_{t_{i+1}}$ increases the rate of U

$$x + x' = X \quad and \quad X > 1 \qquad (7.12)$$

Now in this case based on per-task utilization task is schedulable but overall utilization (U) of processor ρ_j is greater than 1. Hence according to equation (7.3) scheduler is not able to execute overloaded task.

Suppose the overall utilization value of other processor $\rho_{j+1}(\in DS)$ is

$$U = X' \; and \;\; 0 < X' < 1 \qquad (7.13)$$

As we know that ρ_j and ρ_{j+1} processors belong to distributed system DS. In order to balance the load and execute the schedulable tasks, overloaded task (task t_{i+1} increases the value of U) has to migrate towards other processor ρ_{j+1} whose utilization value is less than 1. After the migration of task t_{i+1} utilization values of processors ρ_j *and* ρ_{j+1} has modified now.

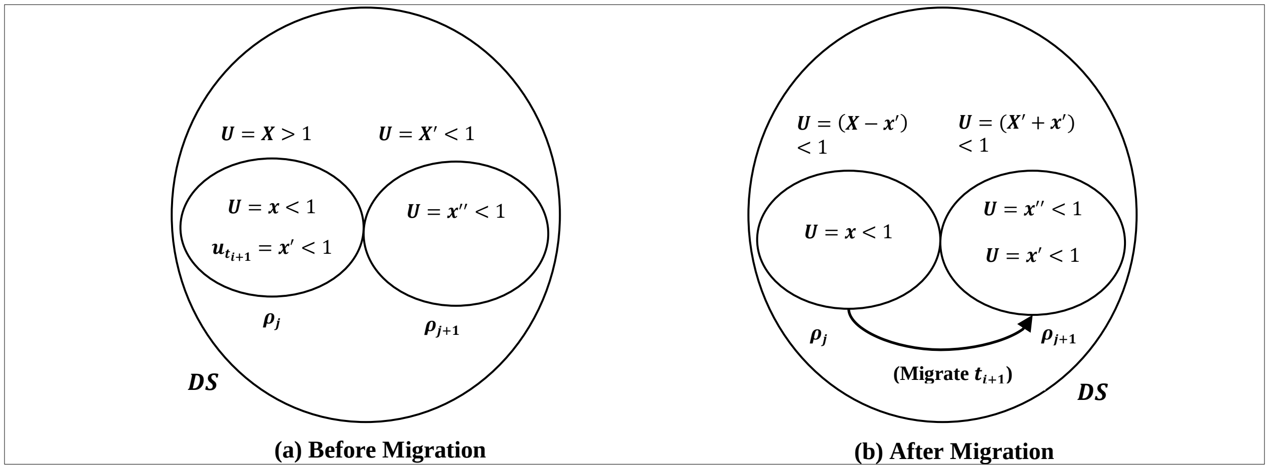


Fig.7.2. Processors scenario after and before Task Migration

7.4.2 *Utilization based Task Migration Algorithm*

In order to implement given algorithm following policies are adopted:

1. **Threshold policy:** Every processor has fix carrying capacity that restricts the total number of task accommodation limit.
2. **Execution Policy :** The execution of task divides into two sections**:**
   a. *Schedulability Test: It checks whether the task is EDF schedulable or not.*
   b. Task Execution: This phase executes task.
3. **Migration Policy:** After the arrival of task, task has to run the acceptance test due to which it has to migrate to other processors because of unavailability of time or utilization capability of its source processor**.** Every processor of the system make its update after fixed amount of time to the main real time scheduler. Moreover, main real time scheduler sorts processors in descending order (in every millisecond) based on their latest utilization value and migrate victim tasks to the processor having least utilization.

Algorithm Earliest Deadline First

```
Input: Random arrival of tasks with $A_i, C_i, P_i, D_i$
Output: Number of tasks meets the deadline (hard), miss the deadline (with
tardiness) with other parameters.
```

```
BEGIN
1. If $u_{t_i} \leq 1$
2.     If  $U \leq 1$
3.         then task is schedulable and assign $taskpriority$ of task on given processor
4.     Else migrate the task
5. Else task is non-schedulable
END
```

7.5 *Entropy Based Task Migration*

7.5.1 *Mathematical Explanation of Proposed task migration algorithm*

Entropy is not a new term we all are aware with it. Every obsession where uncertainty, disorder, confusion, or critical behavior is present entropy concept is applicable. In this chapter, we are just calculating the amount of entropy of the processor or task and use that calculated value in making our RTS stable, balanced and reliable.

Before the calculation of entropy values of given system, first we have rewritten equation (7.8) and (7.10) confirm the presence of entropy in given task or processor.

$$u_{t_i} \quad \begin{Bmatrix} if < 1 & task\ is\ schedulable\ with\ less\ Uncertainty \\ if = 1 & presence\ of\ maximum\ Uncertainty \\ Otherwise & Discard\ the\ task \end{Bmatrix} \tag{7.14}$$

Equation (7.14) tells about the existence of uncertainty after the arrival of task in uniprocessor. The calculation method of amount of uncertainty and its significance will be discussed in further equations.

Similarly, in DS Processor utilization will be

$$U \quad \begin{Bmatrix} if < 1 & tasks\ will\ be\ executable\ with\ less\ Uncertainty \\ if = 1 & presence\ of\ maximum\ Uncertainty \\ Otherwise & tasks\ unable\ to\ meet\ their\ respective\ deadline \end{Bmatrix} \tag{7.15}$$

Hence, when value of U was 1 at that time presence of uncertainty in processor is maximum because within a given amount of time value of U either > 1 (due to arrival of new task) or < 1 (any running task may be complete its execution). In proposing technique, we calculate entropy values of task as well as processor instead of calculating utilization factor.

Equation (7.14) and (7.15) tell us that there is a presence of doubt when a value of per-task utilization or processor utilization reaches to 1. By using entropy concept, author has calculated the presence of uncertainty in given processor.

At this juncture, information theory concept is used for the calculation of entropy values. Because if we merge equation (7.6) and (7.7) then we will get

$$T_i \in \rho_j \in DS \tag{7.16}$$

Arrival and execution of T_i number of tasks are information for processor ρ_j and loads of processors are information for entire RTDS. Hence, in order to analyze the entropy of processor there is a necessity of information about given processor. Therefore, from that amount of information the computation of the amount of disorder or uncertainty will be judged.

Let t_i is some task, which occurs with probability: $Probability(t_i)$. Author has considered that we have received

$$I(t_i) = \log_2 \frac{1}{Probability(t_i)} \tag{7.17}$$

bits of information about missing or meeting of deadline.

Here t_i is real time task that occurs with $\{A_i, C_i, D_i, P_i\}$. Since, time is continuous variable therefore before calculating $Probability\,(t_i)$. Seek to understand following example taken from [AoPS Incorporated and Solving, 2006]:

Example3.1. Lawrence parked his car in a parking lot at a randomly chosen time between 2:30 PM and 4:00 PM. Just half an hour later, he drove his car out of the parking lot. What is the probability that he exited the car park after 4:00 PM?

Solution:

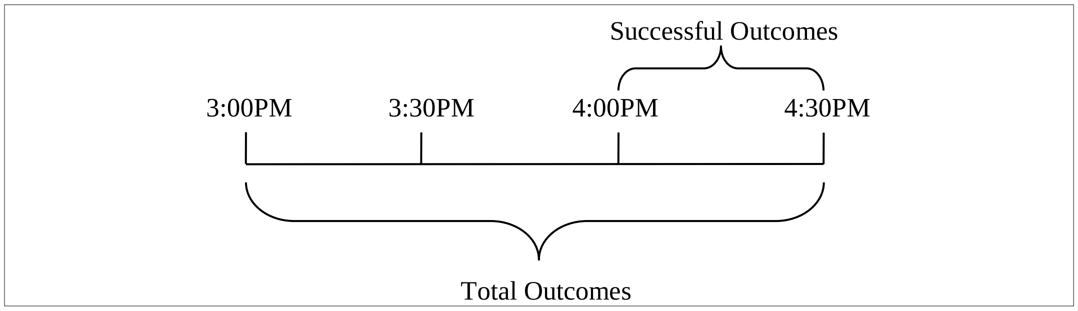


Fig.7.3. Probability of continuous variable

$$Probability(Success) = \frac{length\ of\ successful\ outcomes}{length\ of\ total\ outcomes} = \frac{30(minutes)}{90(minutes)} = \frac{1}{3} \qquad (7.18)$$

Similarly, let us take an example of single task with following details: $A_i = 1{:}00\ PM, C_i = 30\ min, d_i = 60min$

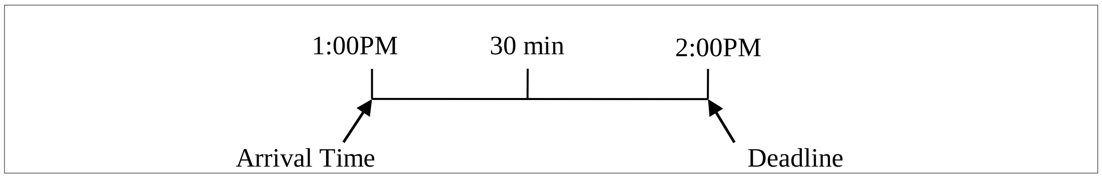


Fig.7.4. Real Time Task scenario

Task can meet the deadline iff its $C_i \leq D_i$. Hence, length of total outcome will be its deadline *i.e.60 minutes* and length of successful outcome must be $A_i + C_i$. Therefore,

$$Probability(meet) = \frac{A_i+C_i}{d_i} = \frac{30(minutes)}{60(minutes)} = \frac{1}{2} \qquad (7.19)$$

and

$$Probability(miss) = 1 - Probability(meet) = 1 - \frac{1}{2} = 0.5 \qquad (7.20)$$

Occurrence of task Arrival contains two outputs meeting and missing of deadline. Every event has information about its output. Hence, $I(t_{imeet})\, and\, I(t_{imiss})$ computes following information:

$$I(t_{imeet}) = \log_2 \frac{1}{Probability(meet)} = \log_2 \frac{1}{0.5} = \log_2 2 = 1\, bits \qquad (7.21)$$

Correspondingly,

$$I(t_{imiss}) = 1\, bits \qquad (7.22)$$

Now, entropy of the given task computes with $P(meet) and\ P(miss)$ probabilities.

$$Entropy\ (t_i) = Probability(meet) \times I(t_{imeet}) + Probability(miss) \times I(t_{imiss}) \qquad (7.23)$$

$$= Probability(meet) \times \log_2 \frac{1}{Probability(meet)} + Probability(miss) \times \log_2 \frac{1}{Probability(miss)} \qquad (7.24)$$

$$= 0.5 \times 1(bits) + 0.5 \times 1(bits) = 1 bits \qquad (7.25)$$

Entropy has following properties [D. Feldman, 2002]:

1. Non-negative value, $Entropy(system) \geq 0$
2. $Entropy(system) = 0\ iff$ system is known with certainty. *i.e.* the probability of one outcome is 1 and probability of all other outcomes is 0
3. Larger the value of $Entropy(system)$, the more helpful, on average, a measurement of system is.

Equation (7.14)-(7.25) is all about the calculation of entropy values of given task as well as processor. Before the discussion of proposed migration algorithm, let us take a brief introduction about the maximum entropy model. The principle of maximum entropy is based on hypothesis that when we calculate the probability distribution, we should pick that distribution which leaves the largest enduring uncertainty (the maximum entropy) [D. Chen, 1998; P. Penfield, 2003; Dong Yu, 2009 & R. Malouf, 2002]. Space and time are two main complexities of a particular processor based on which system has some time boundary restrictions for the processing of allotted processes (task/information). Hence, we define the maximum information carrying capacity (threshold) in the term of maximum entropy of given processor that is based on some standard definition mentioned in previous chapter.

Author has executed a limited number of tasks having some fine granular task with fundamental parameters (Worst Case Execution time, Deadline) on a given processor. Then the maximum entropy of each task is computed and these calculated tasks entropy decides the maximum entropy limit of a particular processor. This fixed maximum entropy limit works as a reference standard ($E_{processor} \leq maxentropy$). Now this reference standard will work as a dynamics-governing factor in our simulation just like utilization factor ($U \leq 1$). Whenever system entropy reaches the maximum entropy limit any further addition to the processor will invoke task migration.

Author has considered the carrying capacity of given processor is 100 tasks (threshold). Each task arrives at the same time by following finely granular fundamental parameters:

WCET= 100millisec and deadline=10000millisec

Equation (7.19) and (7.20) compute the two probabilities of each task. We will get:

$Probability(meet) = .01$

and

$Probability(miss) = 0.99$ *(7.26)*

Entropy values by using above calculated probabilities are:

$Entropy(task_{meet}) = 0.066\ bits$

and

$Entropy(task_{miss}) = 0.014\ bits$ *(7.27)*

As in [S. Shimokawa, 2001] author has observed that we should choose the probability that leaves with higher entropy (or uncertainty). From above calculated entropy values, we get maximum entropy from $probability\ (meet)$ and so it will be our maximum entropy value for a given task.

We are talking about the processor's maximum entropy limit:

$$maxent(processor) = \sum_{i=1}^{100} maxent(task_i) = 6.6\ bits \tag{7.28}$$

In this way, author has decided a maximum entropy limit for a given processor and it depends on maximum carrying capacity of processor that can vary from processor to processor. For load balancing, it works same as utilization based work.

7.5.2 *Entropy Based Algorithm*

Entropy Based Algorithm

```
Input: Periodic arrival of tasks with $A_i, C_i, P_i, D_i$
Output: Number of tasks meets the deadline (hard), miss the deadline (with
tardiness) with other parameters.
```

```
BEGIN
$maxent$ = 6.64
    1. If $Entropy(t_i) \leq maxent$
    2. $Current_{entropy} = Current_{entropy} + Entropy(t_i)$
    3. $Available_{entropy} = maxent - Current_{entropy}$
    4.      If  $Current_{entropy} \leq maxent$
    5.       then task is schedulable and assign $taskpriority$ of task on given
                  processor
    6.      Else migrate($Entropy(t_i)$)
    7. Else task is non-schedulable
```

```
    migrate($Entropy(t_i)$)
    BEGIN
        1. If ($Available_{entropy}(\rho_j) > Available_{entropy}(\rho_i)$)
        2.    then destination = $\rho_j$
        3. Else
        4.    destination = $\rho_i$
    END
END
```

7.6 *Simulation, Results and Discussion*

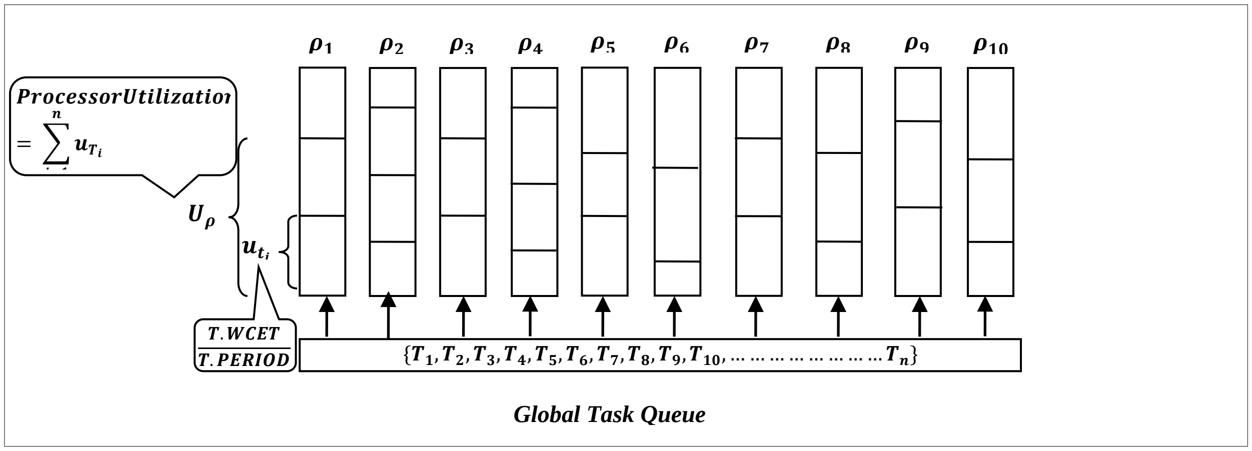
In this section, author presents the simulation, results and discussion. Up-till now the dynamics of any distributed system or real time distributed system are determined by using the utilization factor. Although, utilization factor is working excellent, somewhere we doubt it is the only reasons due to which many tasks are unable to meet the deadline. We utilize the concept of entropy because it is the only factor that evaluates the presence of uncertainty in the performance of tasks. Additionally, it is measured in bits, which is a memory allocation unit in our computer systems. The working of utilization depends only on the time needed by the processor on finishing single task. Thus, we scale up the whole system. Before execution of task, it requires some space/memory in the CPU; for waiting or halt, it also occupies the same space. Based on utilization value scheduler assign tasks to particular processor, but at mean time task miss its deadline. All this happens because scheduler's center of attention (time) is only one side of coin. From our simulation results, we realize that with time, memory is also an important factor in the performance of real time tasks. Pushing memory space to the boundary is not advisable; we must set an accommodating threshold so that extra space/time requirement is always satisfied. Here we have defined fundamental unit of time as well as space. For any activity, we use this unit space and time.

First, author anticipate one-to-one mapping in-between both parameters and surprisingly got it as shown in previous chapter. In order to verify this mapping, normalized entropy of given processor has been mapped with utilization factor. After receiving the results of one-to-one mapping, author simulate RTDS by using well known scheduling algorithm EDF. Then, we put our new parameter entropy in place of utilization and finally got alike or improved results.

At the outset of this section, we first understand the existing scenario of RTDS followed by entropy-based work with experimental set-up. Afterwards, the performance of system will be discussed on the basis of existing and new parameter.

7.6.1 *Existing scenario of RTDS*

Figure (7.5) elucidates the working of RTDS. All real time tasks arrive is maintained by Global task queue of the global scheduler of distributed system. This global scheduler selects processors for task allocation. Every processor has some threshold limit of tasks. In this simulation, no processor will perform more than 10000 tasks (as per the simulation constraint). If tasks cross the utilization limit of a processor than task will migrate to another processor having $U < 1$. The task that preempts number of times from a higher priority task will also become the victim task. How migration of tasks occurs is explained in Figure (7.2). Selection of victim task is framed on the per-task utilization as well as its priority in the priority queue.



ρ_1 ρ_2 ρ_3 ρ_4 ρ_5 ρ_6 ρ_7 ρ_8 ρ_9 ρ_{10}
$ProcessorUtilization = \sum^{n} u_{T_i}$
U_ρ
u_{t_i}
$\frac{T.WCET}{T.PERIOD}$
$\{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}, \dots \dots T_n\}$
Global Task Queue

Fig.7.5. Existing scenario of RTDS

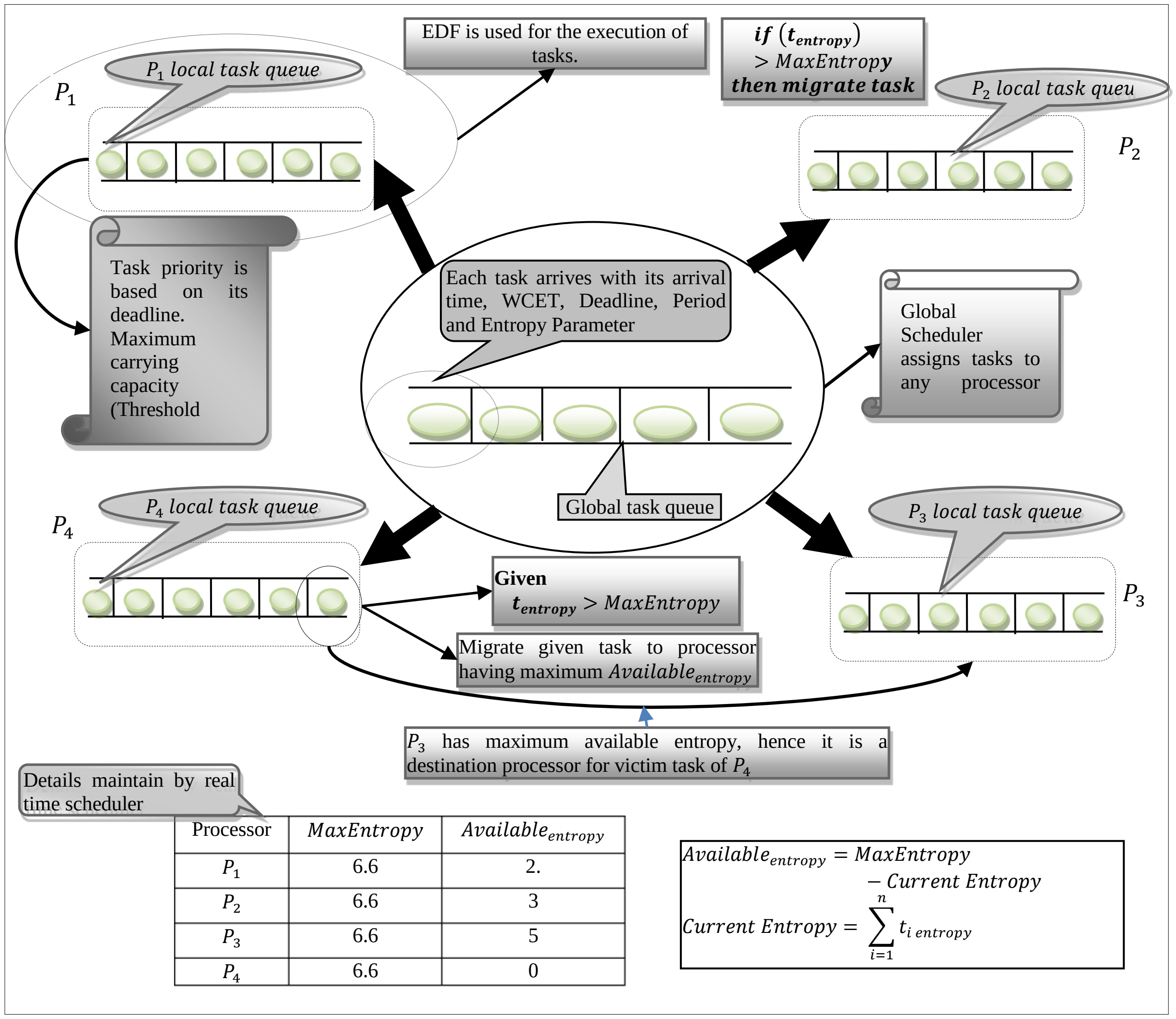
7.6.2 Experimental Set-up and TestBed

For the purpose of implementation and simulation of the algorithm, author has designed a simulator for the execution of real time tasks of RTDS. Figure (7.6) shows the architecture of RTDS that schedule and execute tasks by using a proposed entropy parameter. Figure (7.6) architecture contains following components and technique:

1. ***The Global Task Queue:*** All newly arrived periodic independent real time tasks are arranged in global task queue, which is maintained by a global scheduler of DS. This global scheduler allows task migration from one to another processor. As we know distributed system distribute load to all participated processors of the system. Hence, scheduler assigns tasks of global queue to the randomly selected processor.
2. ***The Local Task Queue:*** Every processor holds this local task queue**.** Execution of tasks follows EDF priority based scheduling algorithm.

3. ***Entropy Values:*** As we have noted in our previous section, that measuring unit of entropy is in bits**.** In computer technology, space is measured in terms of bits. The availability of uncertainty in retrieving information is computed in bits. Hence, $Current\ Entropy$ tells the scheduler that particular task will occupy that much amount of space in processor. Similarly, $Available_{entropy}$ computes amount of vacant space in given processor. Therefore, in order to balance the load of distributed system $Available_{entropy}$ plays a lead role. Processor having maximum $Available_{entropy}$ becomes the destination processor for victim tasks.
4. ***Earliest Deadline First:*** In order to execute the tasks we have used deadline priority based EDF scheduling algorithm.

These techniques and all components are implemented in Eclipse Java EE IDE environment running with Ubuntu Version 11.10, and we periodically generate random independent real time tasks. Java threads and synchronization between them is implemented here for real time tasks generation and execution. We continuously ran upto 10,000 independent real time tasks 30 times on 3, 5, 8 and 10 processors to compute the success ratio, failure ratio, maximum tardiness and efficiency of the entire system.


EDF is used for the execution of tasks.
$if\ (t_{entropy}) > MaxEntropy$ **then migrate task**
P_1 local task queue
P_1
P_2 local task queu
P_2
Task priority is based on its deadline. Maximum carrying capacity (Threshold
Each task arrives with its arrival time, WCET, Deadline, Period and Entropy Parameter
Global Scheduler assigns tasks to any processor
Global task queue
P_4 local task queue
P_4
P_3 local task queue
P_3
Given $t_{entropy} > MaxEntropy$
Migrate given task to processor having maximum $Available_{entropy}$
P_3 has maximum available entropy, hence it is a destination processor for victim task of P_4
Details maintain by real time scheduler

Processor	$MaxEntropy$	$Available_{entropy}$
P_1	6.6	2.
P_2	6.6	3
P_3	6.6	5
P_4	6.6	0

$$Available_{entropy} = MaxEntropy - Current\ Entropy$$

$$Current\ Entropy = \sum_{i=1}^{n} t_{i\ entropy}$$

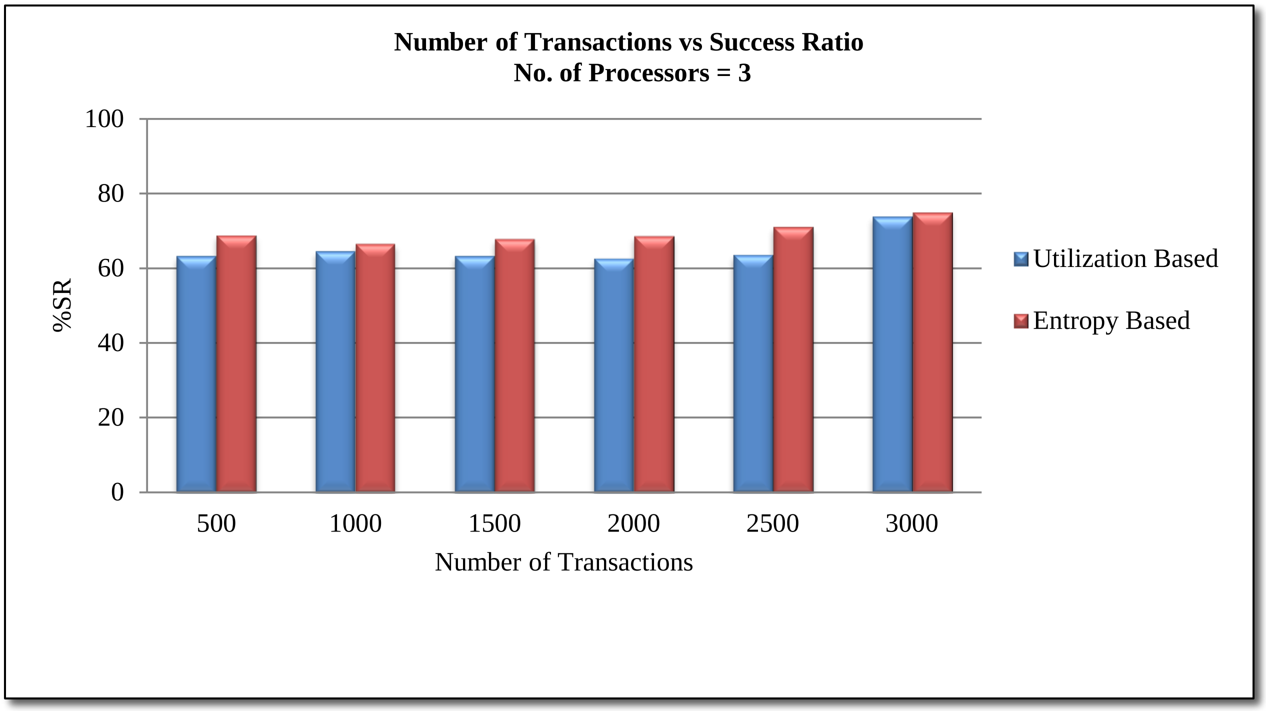
Fig.7.6. Architecture of Real Time task Execution with task Migration (Based on Entropy Parameter)

7.6.3 *Comparison of Dynamics governing parameters*

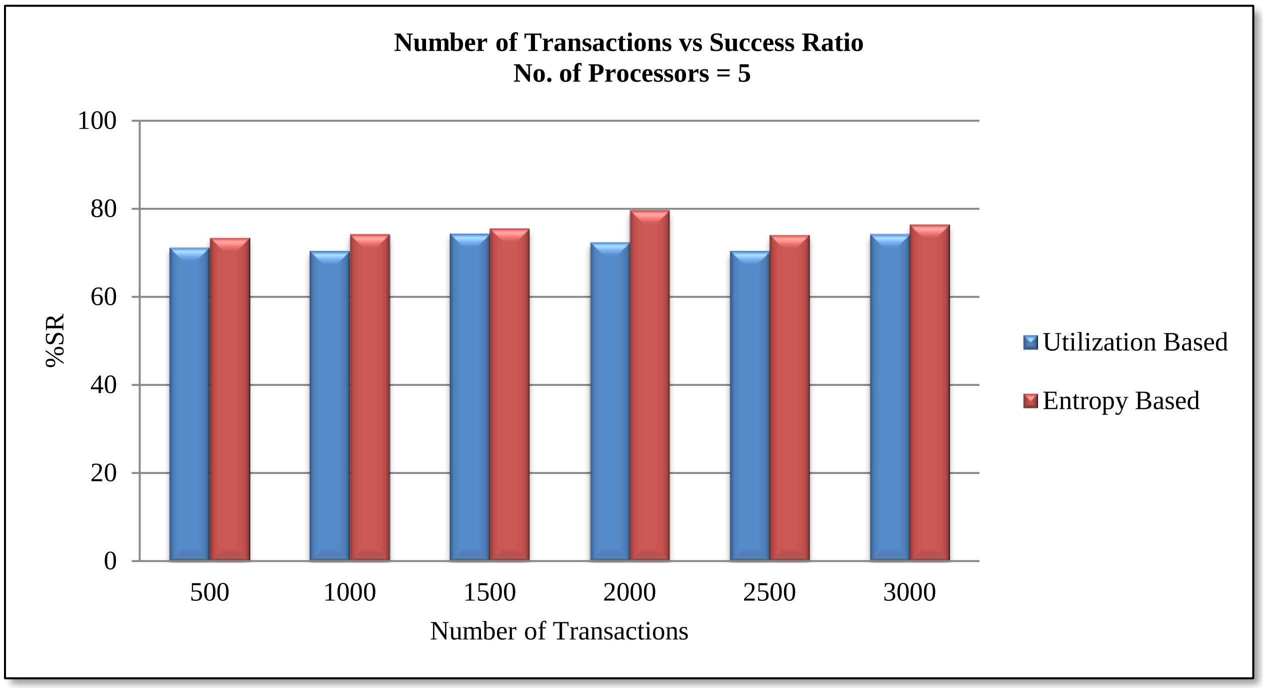
7.6.3.1 *Success Ratio (SR)*

$$SR = \frac{Successfully\ scheduled\ tasks}{Total\ number\ of\ tasks\ arrival} \qquad (7.29)$$

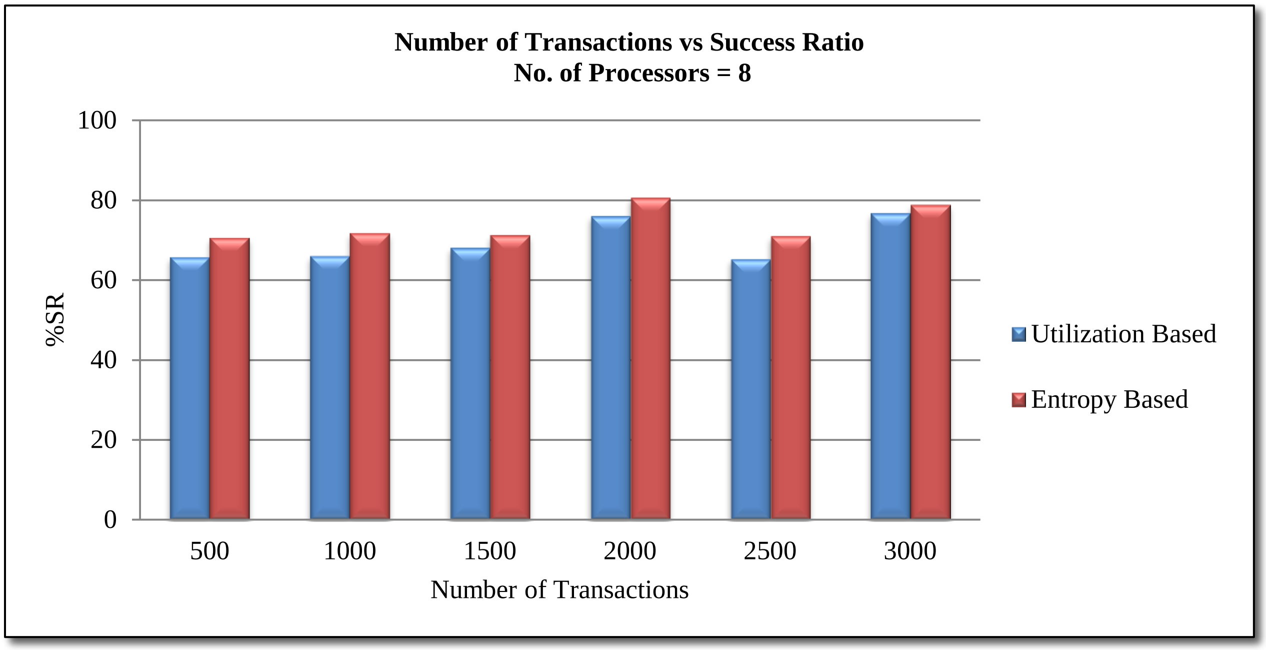
Meeting the deadline is very essential for all real time tasks; therefore, we have computed success ratio that tells the percentage of successfully implemented tasks out of total transactions.



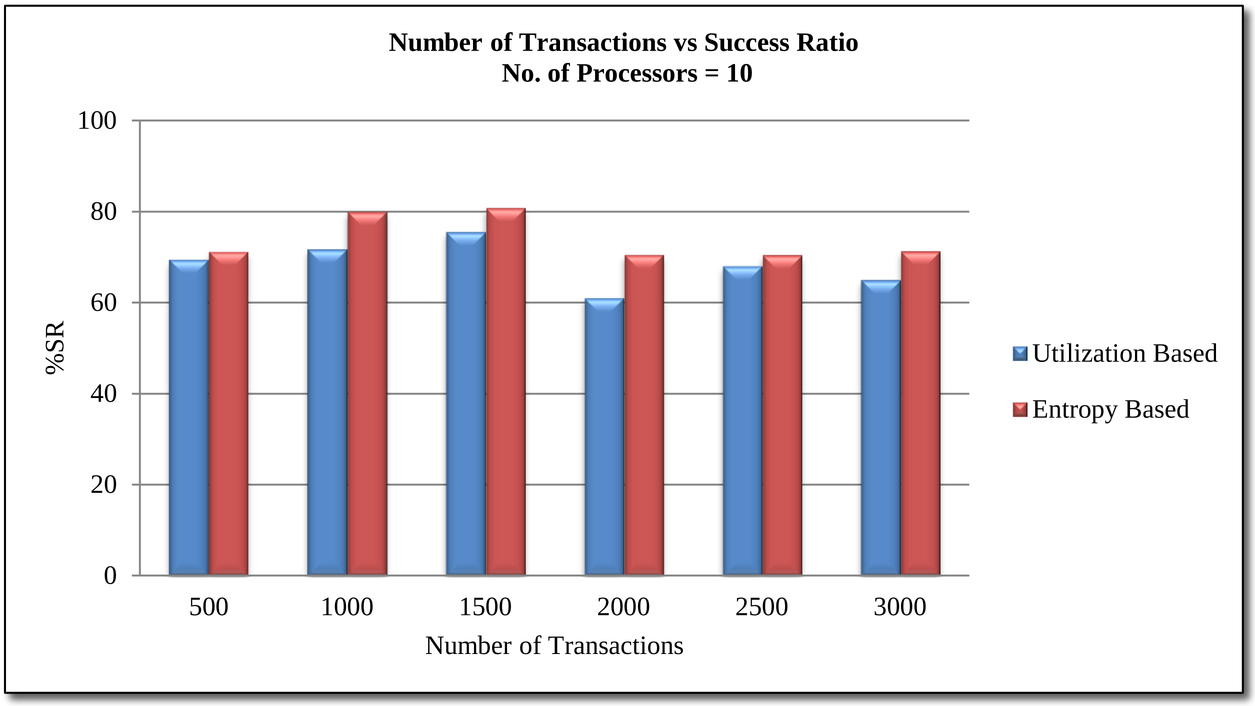
(a) Success Ratio for 3 Processors



(b) Success Ratio for 5 Processors



(c) Success Ratio for 8 Processors



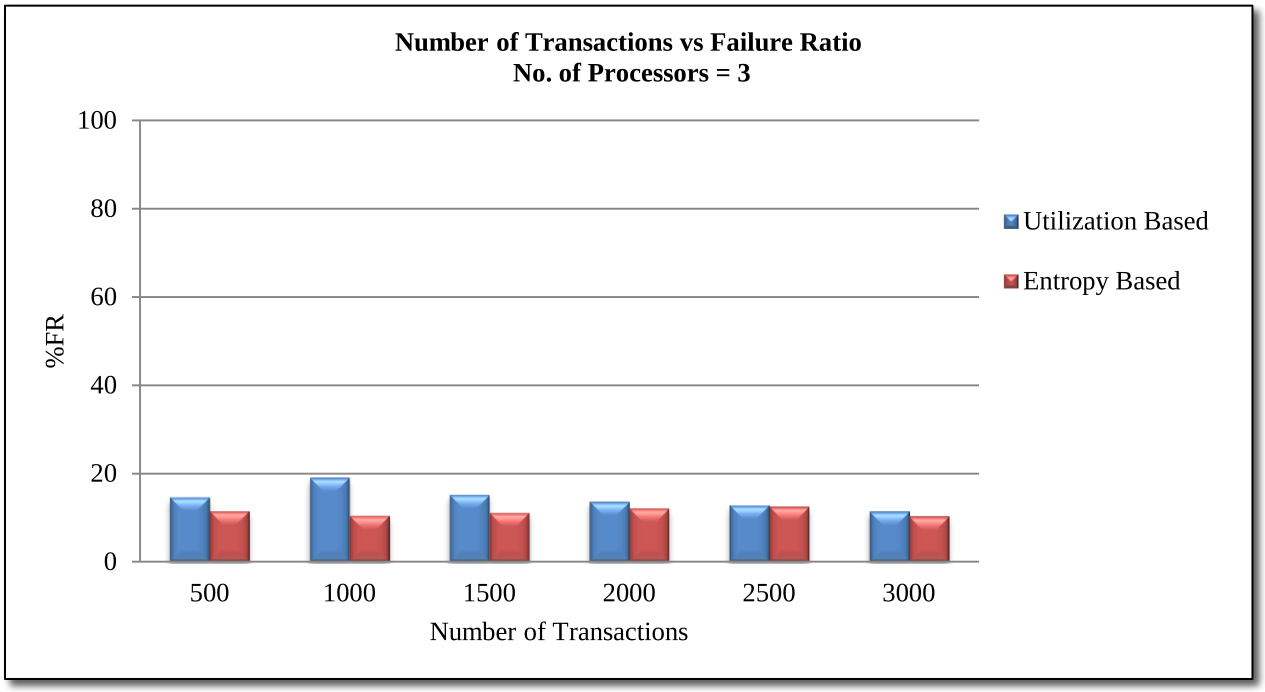
(d) Success Ratio for 10 Processors

Fig.7.7. (a-d) Number of transactions Vs. Success Ratio on 3,5,8 and 10 Processors

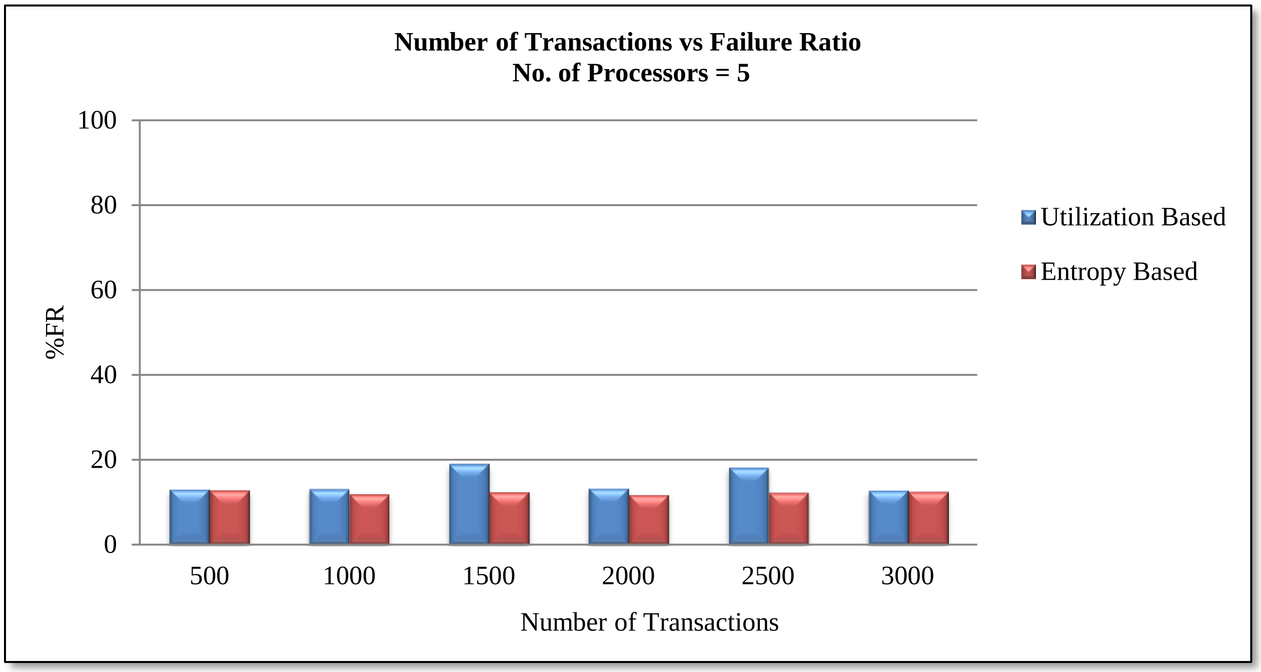
7.6.3.2 *Failure Ratio (FR)*

$$FR = \frac{Tasks\ Miss\ the\ deadline}{Total\ number\ of\ tasks\ arrival} \tag{7.30}$$

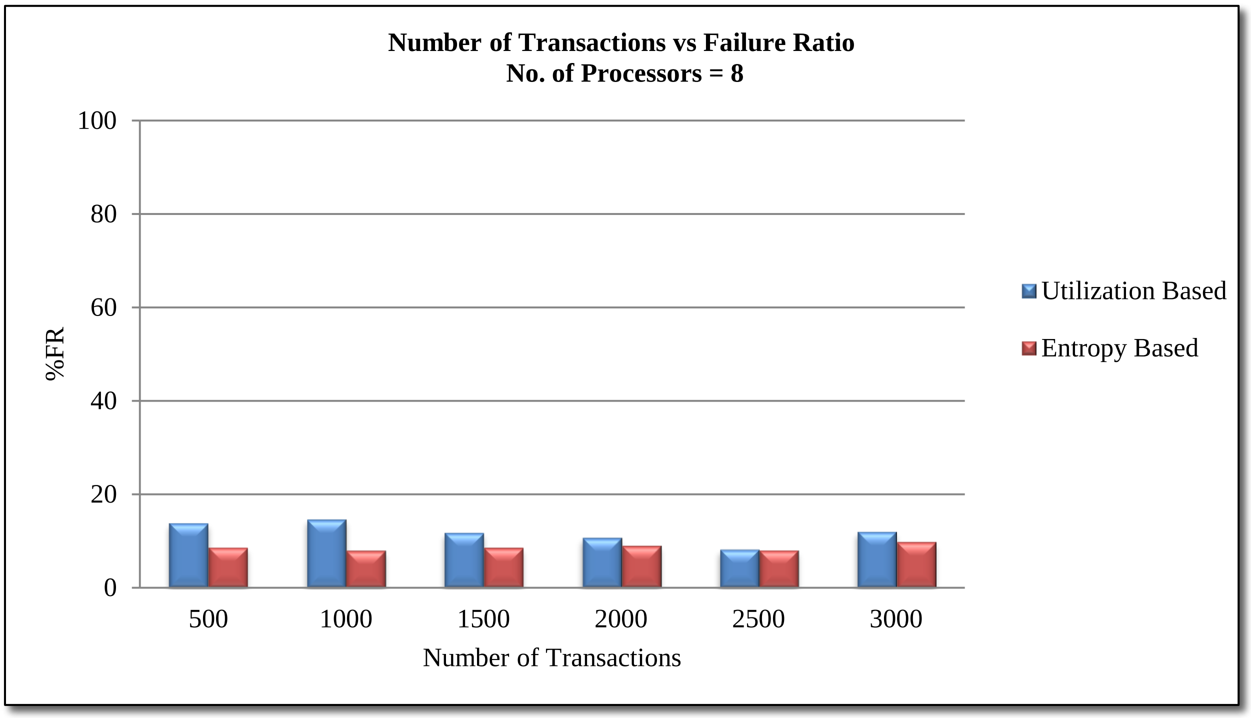
This parameter figure out the percentage of tasks those are unable to meet the deadline. Missing deadline breaks down the system in HRT system and slows down the performance of the system in SRT system. Hence, it is also an arduous task for all algorithms. Hence, failure ratio computation is necessary as well.



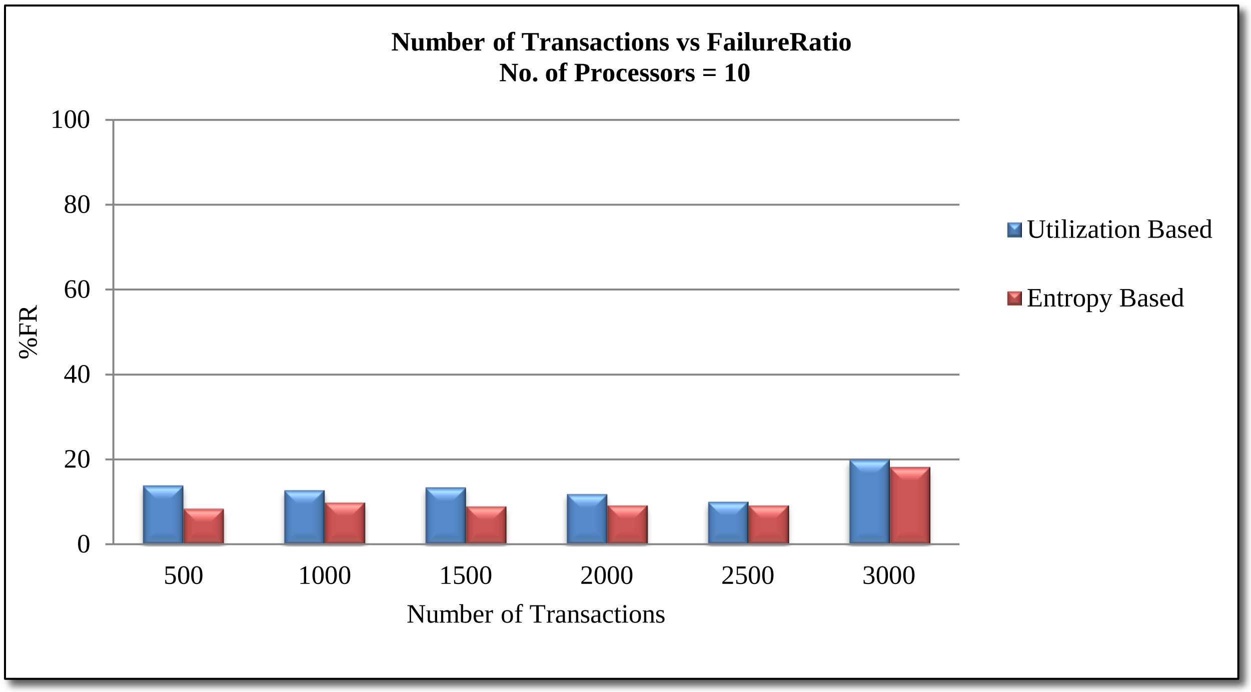
(a) Failure Ratio for 3 Processors



(b) Failure Ratio for 5 Processors



(c) Failure Ratio for 8 Processors



(d) Failure Ratio for 10 Processors

Fig.7.8. (a-d) Number of Transactions vs. Failure Ratio on 3, 5, 8 and 10 processors

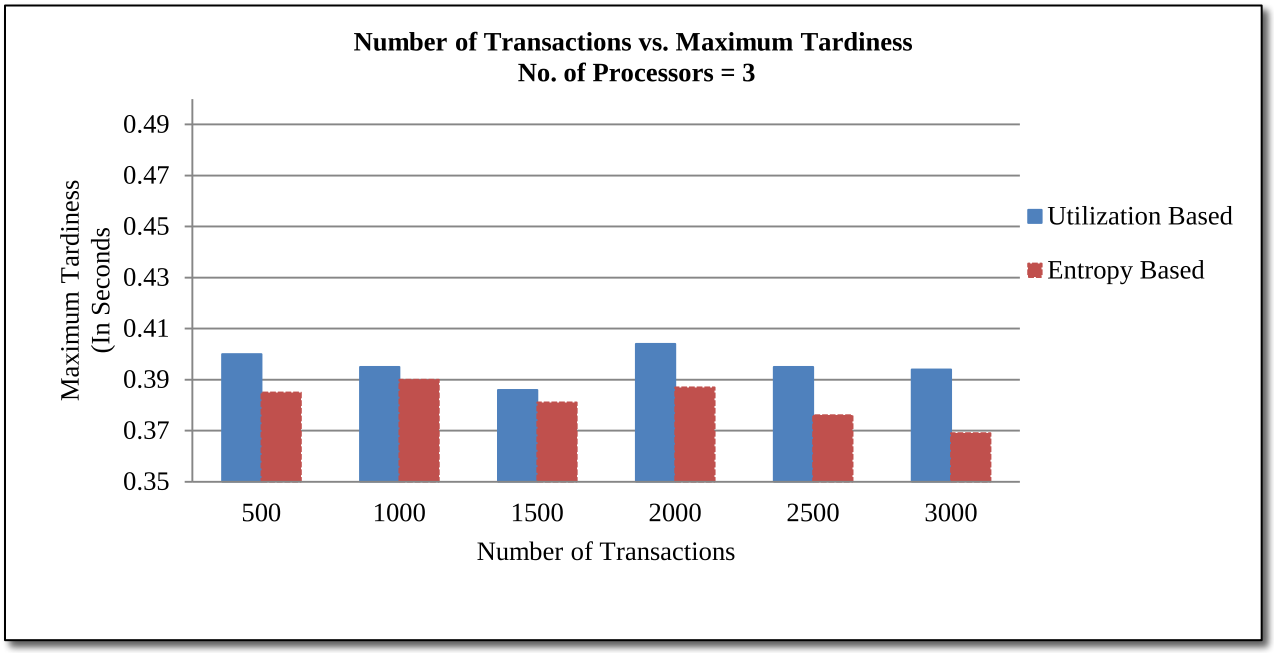
7.6.3.3 *Maximum Tardiness*

As we know that tardiness is the lateness, occur in tasks execution, *i.e.*

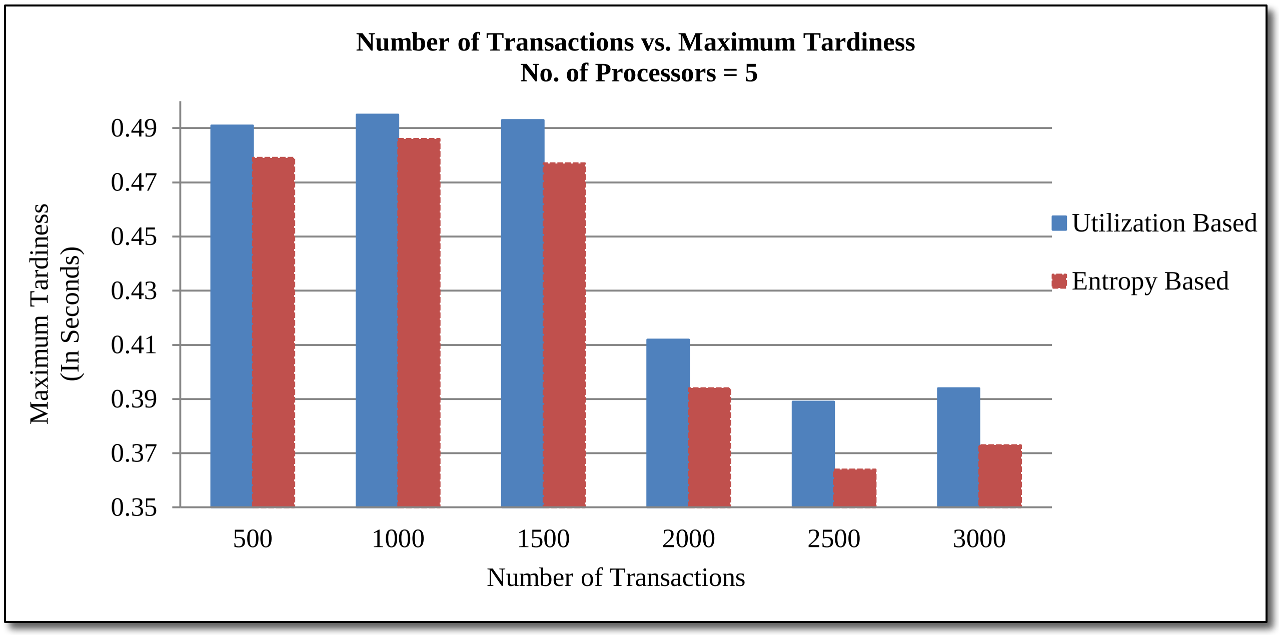
$$tardiness = TST - D_i \qquad (7.31)$$

$$Max\ tardiness = \max(tard_{t_i}) \qquad (7.32)$$

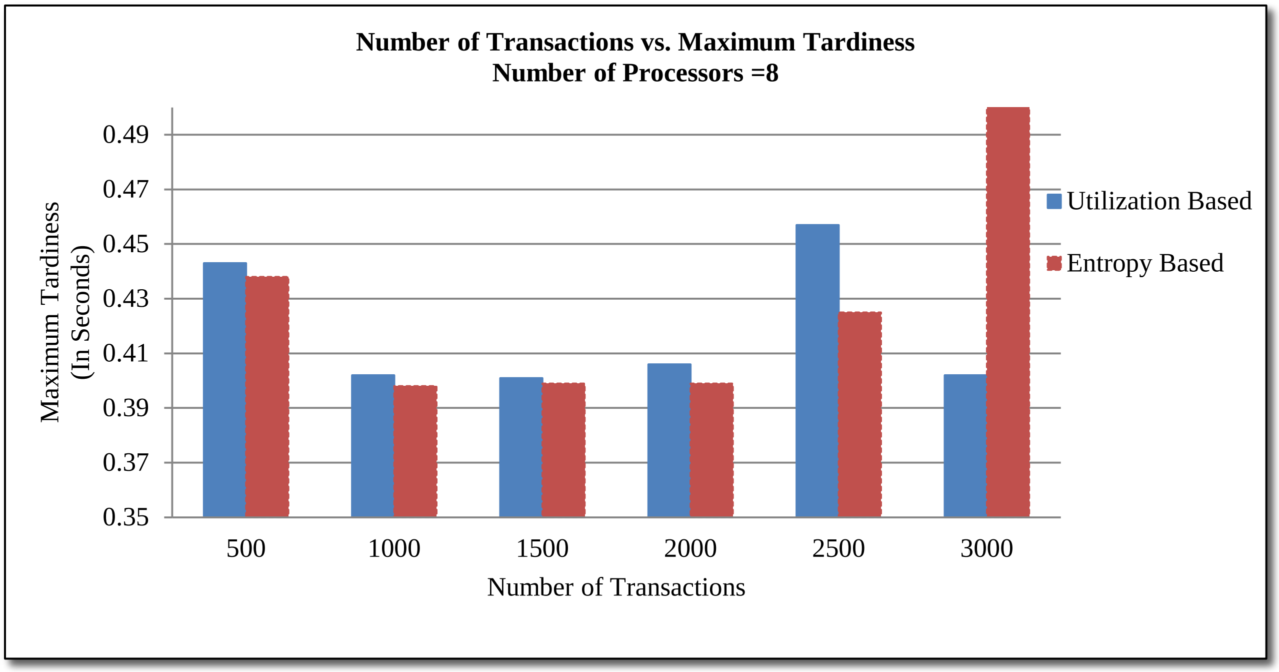
In soft RTS, missing deadline degrades the performance of entire system, therefore tardiness computation is required. Tardiness tells about the performance of the system.



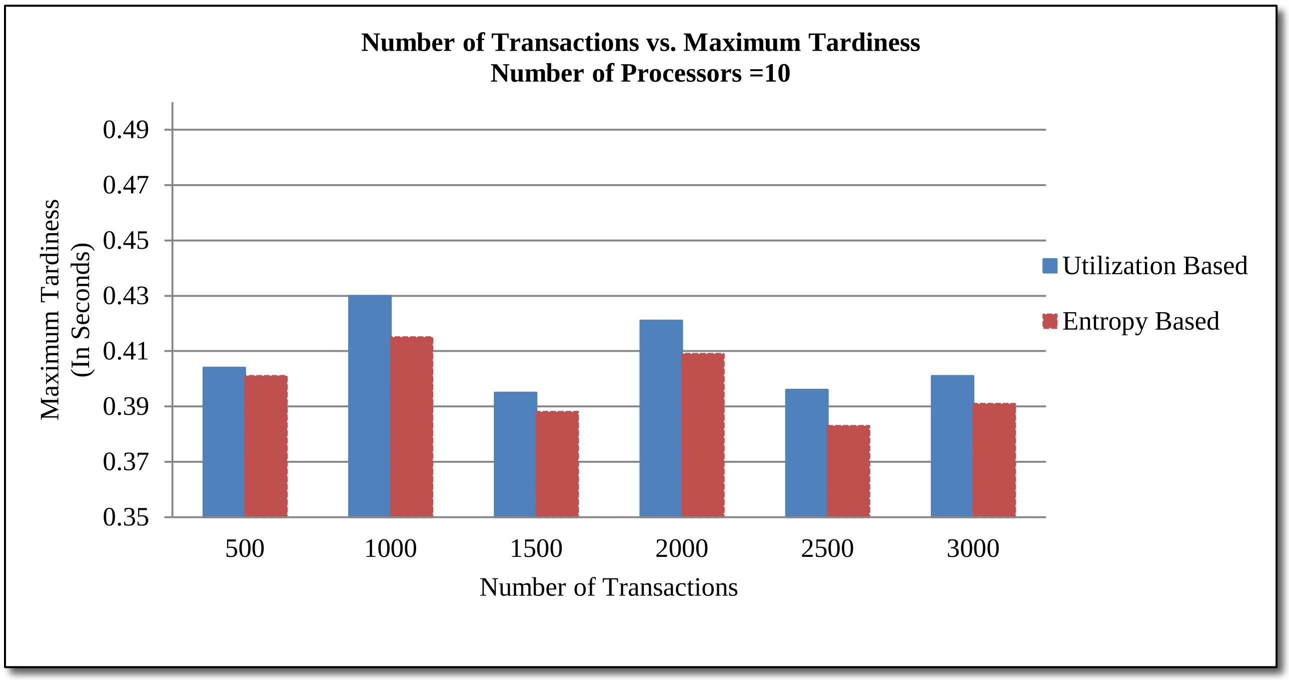
(a) Maximum Tardiness of missed tasks on 3 Processors



(b) Maximum Tardiness of missed tasks on 5 Processors



(c) Maximum Tardiness of missed tasks on 8 Processors



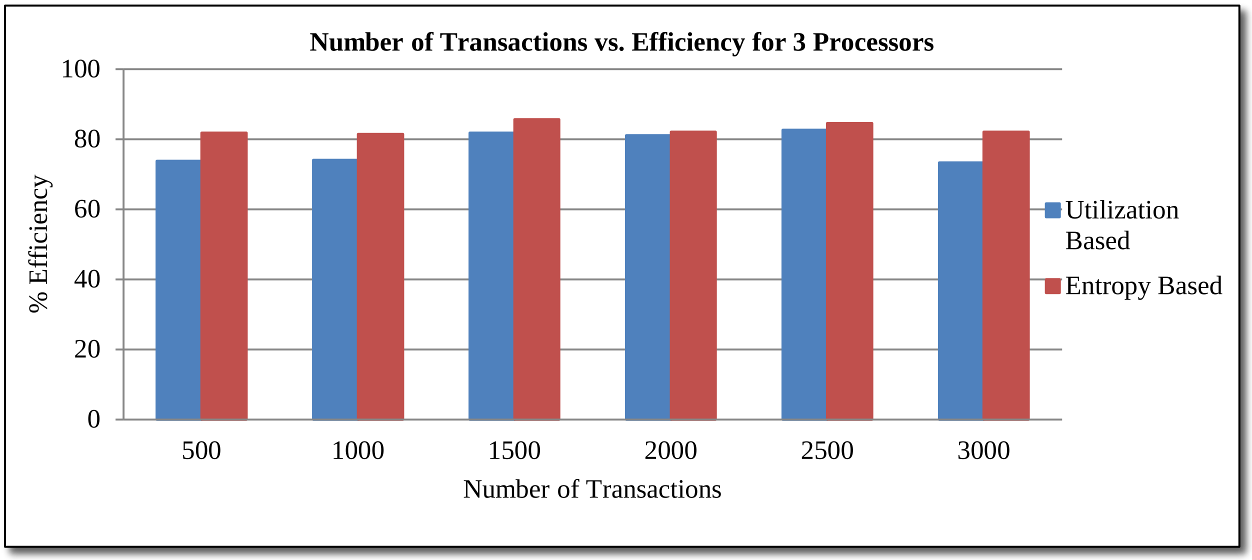
(d) Maximum Tardiness of missed tasks on 10 Processors

Fig.7.9. (a-d) Number of Transactions vs. Maximum Tardiness on 3, 5, 8 and 10 processors

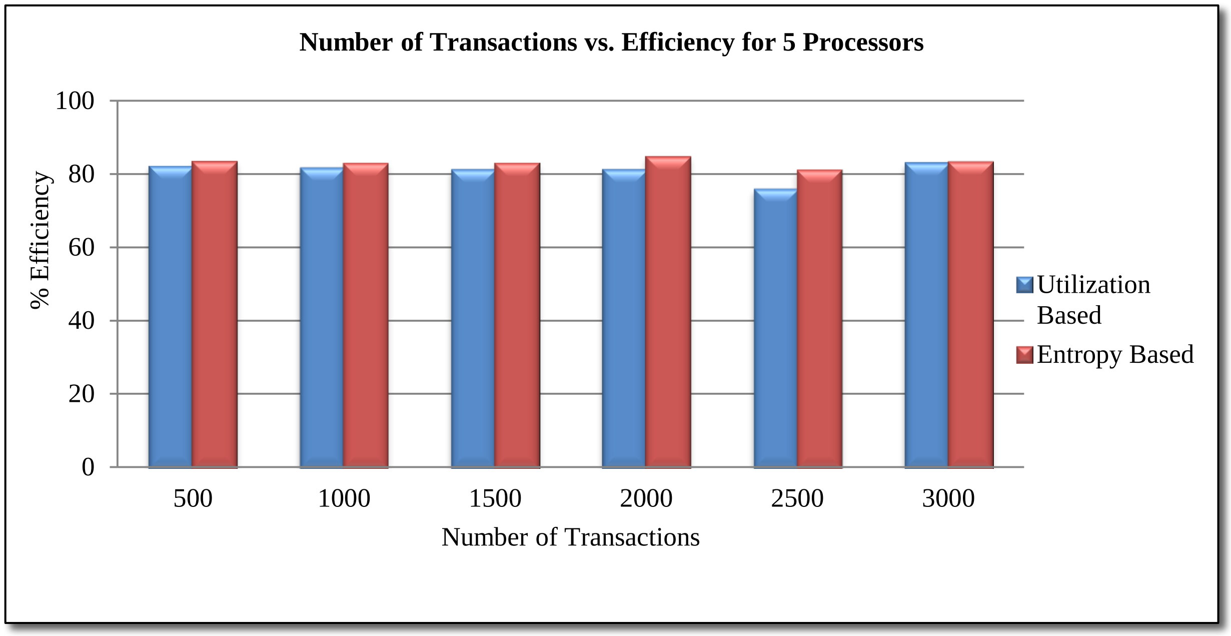
7.6.3.4 *Efficiency*

The average global efficiency of the system is calculated by using:

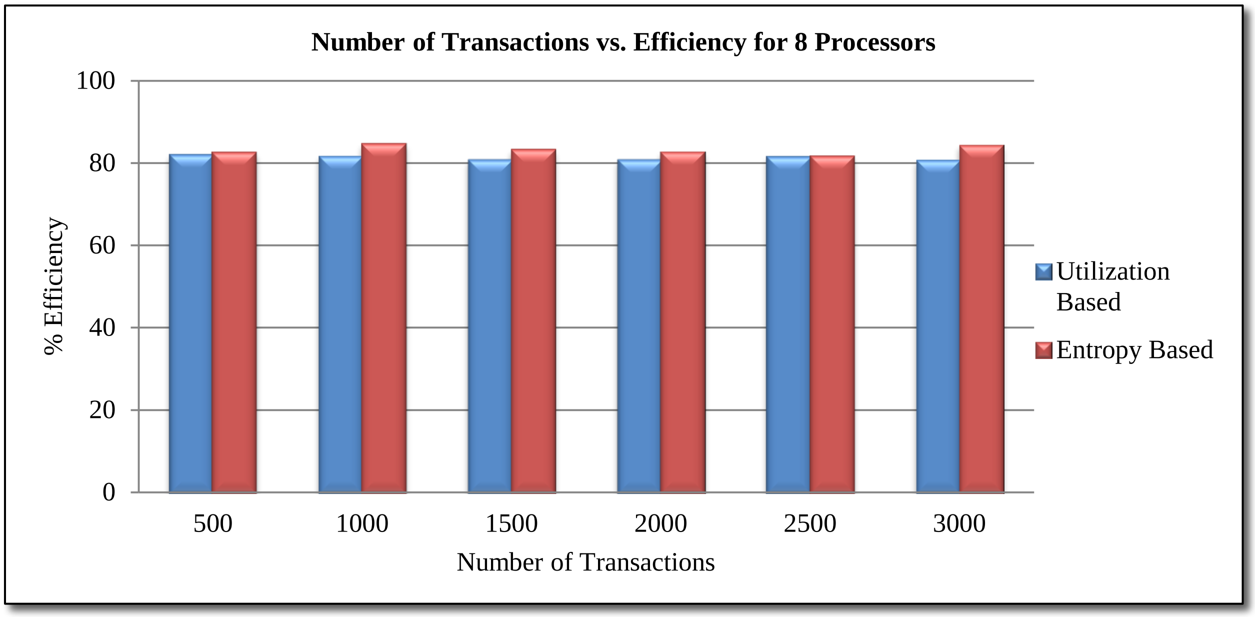
$$\frac{Average_{t_{meet}}}{Average_{t_{miss}}+Average_{t_{meet}}} \times 100 \qquad (7.33)$$



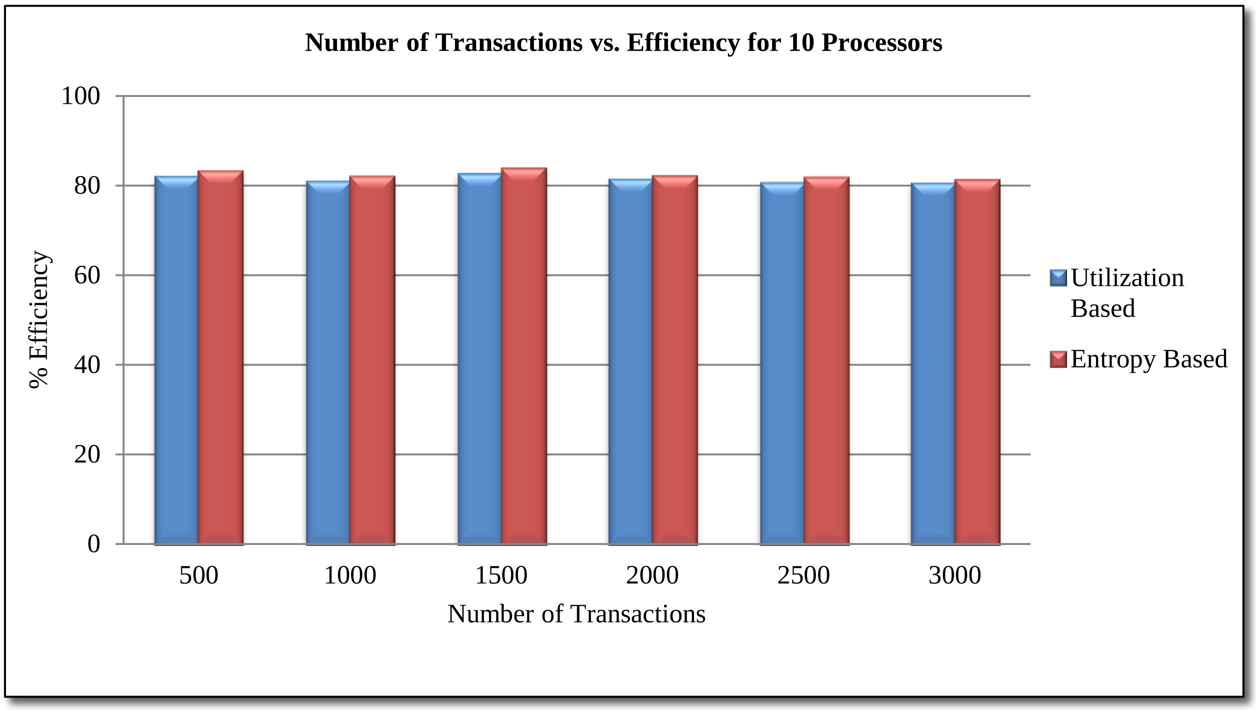
(a) Efficiency of RTDS for 3 Processors



(b) Efficiency of RTDS for 5 Processors



(c) Efficiency of RTDS for 8 Processors



(d) Efficiency of RTDS for 10 Processors

Fig.7.10. (a-d) Efficiency of the system based on Entropy and Utilization

In order to evaluate the performance of entropy based algorithm author run 10,000 independent tasks 30 times on 3, 5, 8 and 10 processors. Results of one-to-one mapping between both parameters encouraged author to replace entropy over utilization. After putting entropy in place of utilization, we get better results. Our performance matrix contains success ratio, failure ratio, maximum tardiness and overall efficiency of the entire system. Graphs of figure (7.10) show that if we carry through the EDF scheduling algorithm by using entropy parameter maximum number of tasks meet the deadline as compared to utilization. Failure of tasks in real time system is very important that tells scheduler about the worst performance of system. The failure ratio of entropy based scheduling is less as compared to existing one (Figure 7.11). Whenever, tasks miss the deadline, its tardiness tells about the lateness in execution. We always compute worst situation therefore, we have computed the maximum tardiness in each case (Figure 7.12). Efficiency of entire system depends on the number of tasks meet and misses the deadline. Figure (7.13) explains the efficiency of a system with 3, 5, 8 and 10 processors and we get better efficiency as compared to existing one. Along with all above discussed parameters, entropy has some additional features.

7.7 *Summary*

In this chapter, author has demonstrated a new dynamics governing parameter entropy to replace utilization factor in RTDS. We have simulated real time environment by using maximum entropy instead of utilization factor. The threshold value of utilization is usually set to one; synonymously here we assume the entropy threshold value to be the maximum entropy for a given processor. Maximum entropy for a processor can be defined as the total amount of entropy present instantaneously when a processor (fully loaded with the task) cannot additionally accommodate new task. Further, we use an entropy threshold in place of utilization factor, calculated average efficiency of the entire system and verified our results with one-to one mapping and then with performance of entire processor. Author is happy with results that the efficiency of both type systems (govern by utilization factor and entropy) independently show similar fluctuation range. Results we obtain are alike or healthier than the existing one. These are just preliminary results but are worth mentioning. Ultimately, we conclude to have another parameter that can regulate the dynamics in RTDS parallel to utilization factor. Here, comparison of entropy based EDF algorithm is compared with utilization based EDF algorithm. In next chapter author has compared their results with RMS algorithm with some additional parameters. In future, author is planning to complicate it by implementing this on DAG. Author has attempted to determine the best-case scenario and direct applications. More advance model will be investigated in the next scope of this new approach.

CHAPTER 8

EVALUATION AND COMPARISON OF LOAD BALANCING IN RTDS USING INFORMATION THEORETIC ENTROPY

In previous chapter, author implements EDF algorithm with the help of entropy in place of utilization. Afterwards compare both results and find entropy based algorithm working better than the existing one. Now, this chapter compare this entropy based EDF with RM scheduling algorithm based on some other parameters. Main benefit and requirement of any distributed system is load balancing that is achieved by using task migration or task duplication methodology. The task migration technique can be used for independent as well as dependent tasks but duplication of tasks specially implement on dependent tasks in order to lessen the execution cost of an entire DAG by reducing the communication costs in-between tasks [R. Sharma, 2011 & A. Bestavros, 1996].

8.1 An Overview on CPU Utilization

This chapter exploits Information theoretic entropy concept for load balancing in loosely coupled distributed system. Till now processor utilization is the only parameter that plays the vital role in load balancing. Many years back some researchers state that CPU load is efficient for load balancing as compared to CPU utilization [D. Ferrai, 1987]. The cause behind CPU load did better is possible because when a host is heavily loaded, its CPU utilization is expected to be nearly 100% and it is unable to reveal the exact load level of the utilization. In contrast, CPU queue lengths can directly reflect the amount of load on a processor.

By the passage of time, techniques for load balancing have been improved. Now, researchers start working on CPU utilization instead of load. Mostly scheduling algorithms use CPU utilization for the same purpose of loosely coupled RTDS. However, here author follows the older concept of CPU load with some extra zest *i.e.* entropy. Simply dissimilarity between the CPU load and entropy is that the later one is computed by gathering the information from previous one. Further, computed values are used to determine the entropy values that tell about the presence of uncertainty (improbability) in retrieved information. Before giving the computation methods of entropy, let us talk about load balancing with the help of CPU utilization and CPU entropy

8.1.1 *Load Balancing*

Load balancing is the advantage of any distributed system or we can say that without load balancing distributed system is worthless. Load balancing itself is a self-descriptive term that regularly distributes the load among nodes/processors by the migration of tasks of heavily loaded or utilized processors (Source) to lightly loaded or less utilized processors (destination/target). The tasks that migrate in-between the nodes are known as victim tasks. Load among the processors are balanced by using task migration or task duplication techniques [N. W. Fisher, 2007; X. Wang, 2005; X. Wang, 2007 & A. Srinivasan, 2003]. CPU load and CPU utilization are some parameters that help in taking the decision for selection of target processor for migration or duplication of victim tasks. But in this chapter, we are using a parameter CPU Entropy instead of load and utilization. Previous chapters already explain the reason behind using entropy in place of utilization but here we are using entropy for balancing the load of processors.

8.1.2 *Load Balancing and Utilization*

As we know that CPU utilization is the time taken by CPU to execute the given task. If processor’s utilization is near about 100% then it will be a source processor and processor having least utilization becomes the target processor. Based on such judgment of utilization, researchers design many tools and scheduling algorithms that execute real time tasks of distributed system. EDF-fm, RealtssMP tool etc. [A. D. Ramírez, 2012; J. Singh, 2012; M. Bertogna, 2009; J. Anderson, 2005; J. Anderson, 2008; P. Emberson, 2007; J. Goossens, 1999; B. T. Akgün, 1996 & D. R. Cheriton, 1988] has been used CPU utilization for the scheduling as well as migration of tasks.

8.1.3 *Load Balancing and Entropy*

Although, the conception of entropy initially a thermodynamic construct, it has been customized in the other fields of study together with information theory, NLP, image processing, thermo-economics/ecological-economics and evolution. All these are few interdisciplinary applications of entropy [C. Lu, 2004; W. H. Zurek, 1989 & R. Malouf, 2002]. This chapter uses information theoretic based entropy for managing the load of participant processes of loosely coupled distributed system.

The arrival and execution of periodic tasks generate arbitrariness in the system. The term entropy is used to calculate the amount of uncertainty of a particular processor. Less uncertain processor becomes the destination and most uncertain will be a source of victim task. With the help of maximum entropy model ($maxe$) [P. Penfield, 2003& S. F. Gull, 1984] scheduler decides the threshold limit of entropy ($E <= maxe$) that behaves like the maximum limit of utilization *i.e.* ($U <= 100\%$). The computation of maximum entropy value is explained in next section.

8.2 *Proposed Entropy based load balancing scheduling algorithm*

This chapter explains the load balancing with the help of information theoretic entropy concept. This entropy perception follows the CPU load concept. It uses the information of task to compute entropy values of task and processor as well. Basic difference between entropy and utilization parameter is that the entropy deals with space and time but utilization deals with time only [R. Sharma, 2013]. Load Balancing is done by computing the available entropy of participant processors of RTDS. Next section describes an overview of entropy and maximum entropy computation.

8.2.1 *Entropy Computation*

Entropy computes the amount of uncertainty present in the given information. Amount of information is figured by using the occurrence probability of particular events. Any real time task has two events missing and meeting of deadlines. The presence of amount of information about the meeting and missing a deadline is computed by evaluating the probability of occurrence of these two events.

$$\Pr(\tau_{meet}) = \frac{\tau_{arrival}+\tau_{wcet}}{\tau_{deadline}} \qquad (8.1)$$

$$\Pr(\tau_{miss}) = 1 - \Pr(\tau_{meet}) \qquad (8.2)$$

With the help of equation (8.1) and (8.2), amount of information about meeting and missing of deadline will be computed in equation (8.3) and (8.4).

$$I(\tau_{meet}) = \log_2 \frac{1}{\Pr(\tau_{meet})} \qquad (8.3)$$

$$I(\tau_{miss}) = \log_2 \frac{1}{\Pr(\tau_{miss})} \qquad (8.4)$$

Equations (8.5) and (8.6) are evaluating the amount of uncertainty present in retrieved information from (8.3) and (8.4).

$$\tau_{meet_{entropy}} = \Pr(\tau_{meet}) \times I(\tau_{meet}) \qquad (8.5)$$

$$\tau_{miss_{entropy}} = \Pr(\tau_{miss}) \times I(\tau_{miss}) \qquad (8.6)$$

Further equation (8.7) is used to evaluate the total entropy of entire task.

$$\tau_{entropy} = \Pr(\tau_{meet}) \times I(\tau_{meet}) + \Pr(\tau_{miss}) \times I(\tau_{miss}) \qquad (8.7)$$

Now, equation (8.8) computes the entropy of particular processor by the accumulation of entropy values of all available tasks on a given processor.

$$E = \sum_{i=1}^{n} \tau_{i_{entropy}} \qquad (8.8)$$

8.2.2 *Maximum Entropy Computation*

In order to compute the maximum entropy of system, we should choose the probability that gives higher values from available entropies of all events [P. Penfield, 2003]. Hence, equation (8.5) and (8.6) computes the entropy of all events and equation (8.9) and (8.10) returns the maximum entropy valued event. These computed maximum entropy is used to decide the maximum entropy value of entire system.

From equation (8.5) and (8.6)

If $(\tau_{meet_{entropy}} > \tau_{miss_{entropy}})$

$$\tau_{maxe} = \tau_{meet_{entropy}} \tag{8.9}$$

Else

$$\tau_{maxe} = \tau_{miss_{entropy}} \tag{8.10}$$

Equation (8.11) is used to determine the maximum entropy value of a particular processor. For deciding the maximum entropy value of any processor, we have taken some fundamental definitions of parameters of given task (discussed in previous chapters). Consider the threshold limit of given processor is 200 tasks. For 200 tasks maximum entropy will be

$$cpu_{maxe} = \sum_{i=1}^{200} \tau_{i_{maxe}} = 19.9 \tag{8.11}$$

Following table is showing the values of maximum entropy on different threshold limit.

Table8.1. Maximum Entropy Values and CPU Maximum Utilization with respect to threshold limit

CPU Threshold Limit	CPU Maximum Entropy
200	19.9
400	66.4
600	139.5
800	239.1
1000	365.4

8.2.3 *Available Entropy Computation*

The measurement unit of entropy is in bits [D.Feldman, 2002], which is used to evaluate the volume of space present in the CPU memory. Every task requires some space for allocation and then time for execution. Therefore, available entropy gives scheduler the existing vacant space in the memory of a given processor. Equation (8.12) is used to evaluate the available space in the CPU.

$$cpu_{available\ entropy} = cpu_{maxe} - E \tag{8.12}$$

This computed available entropy plays main role in balancing the load in-between processors of the RTDS. Following is the algorithm that is used to balance the load by using entropy values.

ALGORITHM 8.1 *ENTROPY BASED LOAD BALANCING*

INPUT: Periodic generation of independent tasks with its $\tau_{arrival}, \tau_{wcet}, \tau_{deadline}$ and τ_{period}
OUTPUT: Execution of tasks with A_{SL}, D_{MR}, $MigR$ and Execution Ratio of total tasks

```
BEGIN
1.  $maxe = 19.9$ // For 200 independent tasks on each processor
2.  $CPU\ Entropy = CPU\ Entropy + Task\ Entropy$
3.  $Available\ CPU\ Entropy = maxe - CPU\ Entropy$
4.  $\boldsymbol{If}\ (CPU\ Entropy \leq maxe)$
5.          $Task\ is\ Executable\ on\ source\ processor$
6.  $\boldsymbol{Else}$
7.          $Check\ (Available\ CPU\ Entropy\ of\ all\ Processors)$
8.  $Migration\ of\ victim\ task\ on\ Processor\ of\ maximum\ available\ CPU\ Entropy$
END
```

Above stated algorithm 8.1 and complete simulation is implemented in Eclipse Java EE IDE environment running with Ubuntu Version 11.10. Java threads and synchronization functions are implemented here for the generation, execution or synchronization between periodic independent real time tasks. Following are some functions that are used to execute the entire simulation.

Table8.2. Functions used in simulation and their responsibilities

Functions	Responsibility
rand.nextInt(10)+1	Random generation of $\tau_{wcet}, \tau_{deadline}$.
c.get(GregorianCalendar.MILLISECOND)	Generate attributes of task in millisecond
(Math.log(1/P1)/Math.log(2))	Log function uses to compute the amount of information as well as entropy value
Thread.sleep(10)	Generate tasks in every 10 milliseconds
starttime = System.nanoTime()	Compute the start time of destination searching
estimatedTime = System.nanoTime() – starttime	Compute total time of destination searching

list.addLast(task)	Add tasks in a local task queue of a given processor
Thread.currentThread().join(10)	This function is used to maintain the synchronization between tasks
Thread.currentThread().interrupt()	Function is used to stop the execution of tasks

8.3 Results and Discussion

EDF and RMS [J. Anderson, 2005 & B. T. Akgün, 2001] are aged and matured scheduling algorithms that are used for the execution of real time tasks. In current scenario, load balancing is done by using utilization parameter only. We consider here a loosely coupled RTDS, and use EDF, RMS and proposed entropy based scheduling algorithms for the execution of tasks with load balancing. Further, the performance of existing and proposed scheduling algorithms will be evaluated on the basis of certain test parameters followed by discussion.

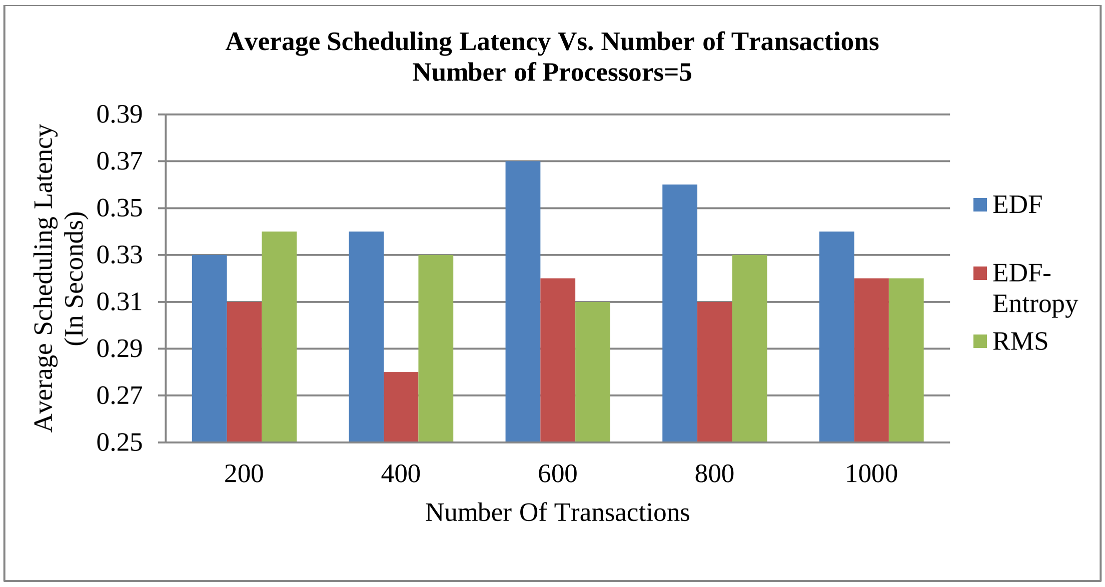
8.3.1 Performance Evaluation

We asymptotically ran up to 5,000 independent real time tasks 30 times for 5 and 10 processors and took readings to compute the scheduling latency, deadline missing rate, migration rate and execution ratio of total tasks.

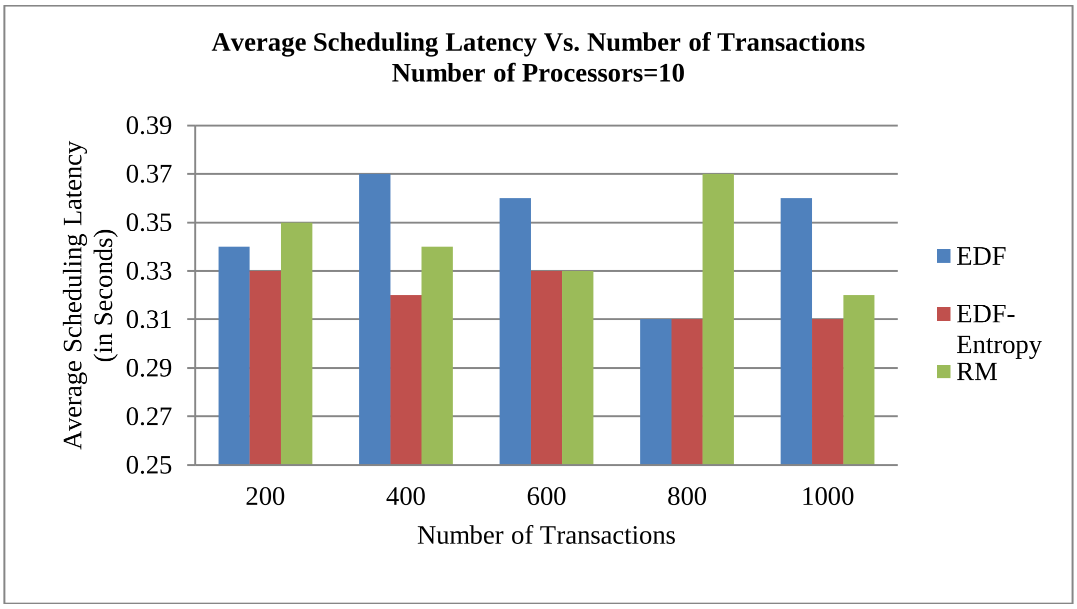
8.3.1.1 *Scheduling Latency(A_{SL})*

Scheduling latency is the time when the system is unproductive because of scheduling tasks. It is a system latency incurred because it has to spend time scheduling.

$$A_{SL} = \frac{Tasks\ with\ scheduling\ latency}{Total\ number\ of\ tasks} \tag{8.13}$$



(a) A_{SL} for 5 Processors



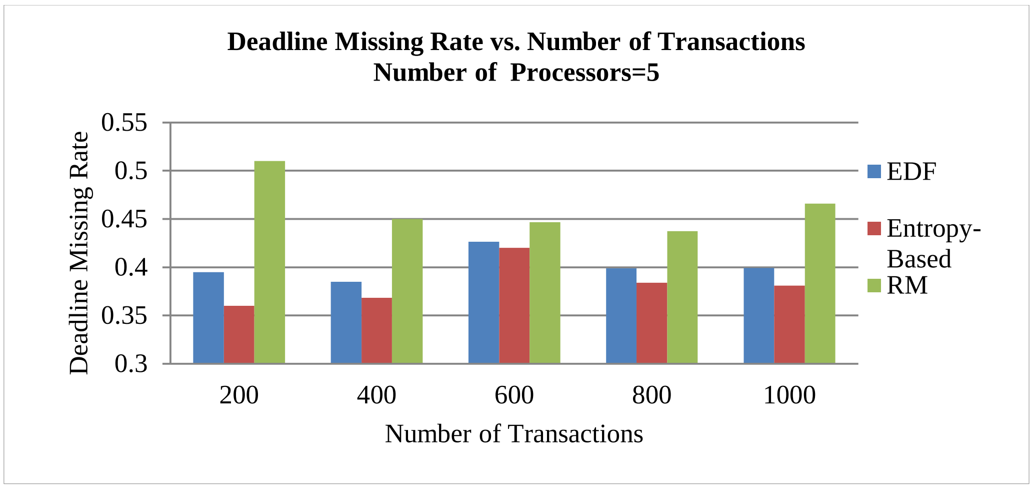
(b) A_{SL} for 10 Processors

Fig.8. 1. Performance of EDF, RM and Proposed Algorithm on A_{SL} for (a) 5 and (b) 10 processors

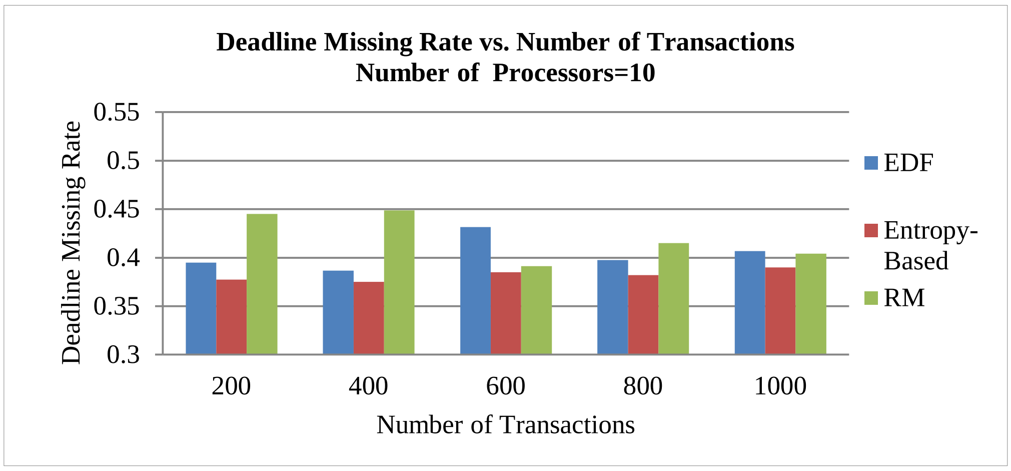
8.3.1.2 *Deadline missing rate (D_{MR})*

This parameter calculates the number of tasks missing deadline per total number of tasks generated at that period.

$$D_{MR} = \frac{Tasks\ missing\ Deadline}{Total\ Number\ of\ tasks} \tag{8.14}$$



(a) D_{MR} for 5 Processors



(b) D_{MR} for 10 Processors

Fig.8. 2. Performance of EDF, RM and Proposed Algorithm on D_{MR} for (a) 5 and (b) 10 processors

8.3.1.3 *Migration rate (MigR)*

Number of tasks migrates per total number of tasks generated has been computed in this parameter.

$$MigR = \frac{Number\ of\ tasks\ Migrate}{Total\ Number\ of\ tasks} \qquad (8.15)$$

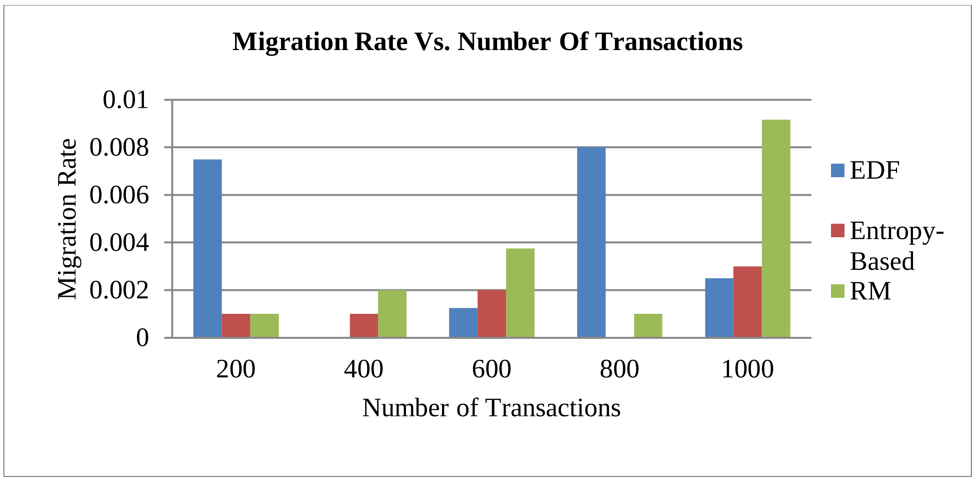
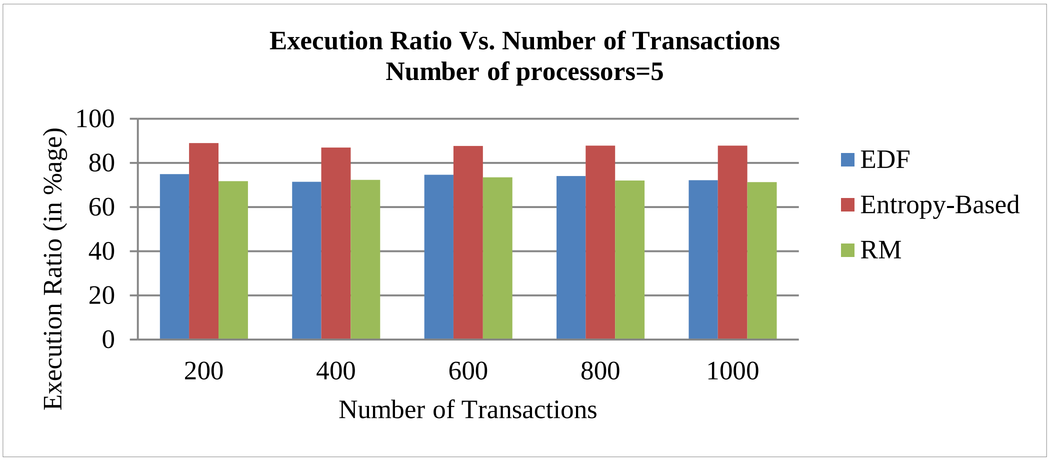


Fig.8. 3. Performance of EDF, RM and Proposed Algorithm based on $MigR$

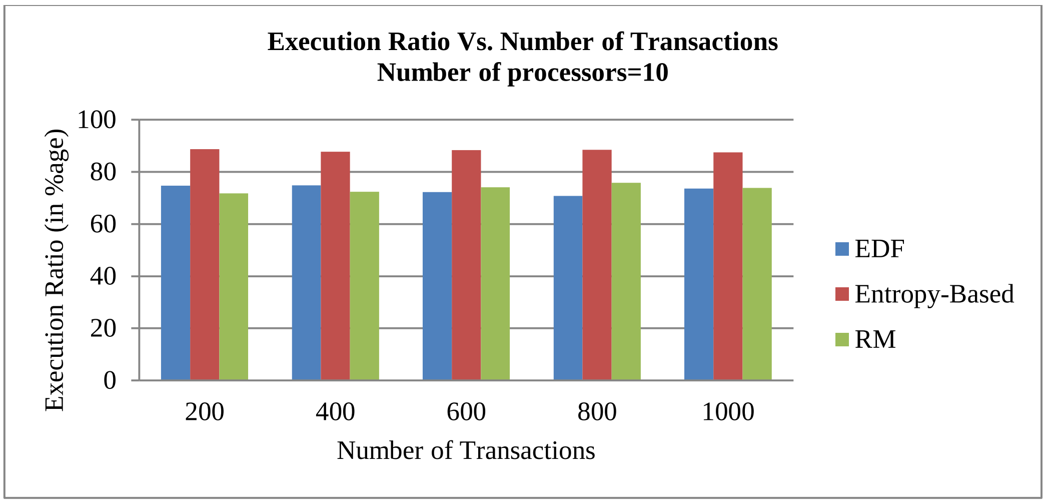
8.3.1.4 *Execution Ratio*

Under the proposed schedulability test, only those tasks are schedulable whose $\tau_{entropy} \leq cpu_{maxe}$, in EDF $\frac{\tau_{wcet}}{\tau_{period}} \leq 1$ and in RM $\frac{\tau_{wcet}}{\tau_{period}} \leq n(2^{\frac{1}{n}} - 1)$. After scheduling of tasks, these tasks will be ready for the execution. Hence,

$$Execution\ Ratio = \frac{Number\ of\ executable\ tasks}{Total\ Number\ of\ tasks} \times 100 \qquad (8.16)$$



(a) Execution Ratio for 5 Processors



(b) Execution Ratio for 10 Processors

Fig.8. 4. Performance of EDF, RM and Proposed Algorithm based on the Execution Ratio for (a) 5 and (b) 10 Processors

8.3.2 *Discussion*

In order to evaluate the performance of information theoretic entropy parameter 5,000 independent tasks run up to 30 times for 5 and 10 processors. Our performance matrix contains scheduling latency, deadline-missing rate, migration rate and execution ratio of total tasks arrived in the system. Graphs of figure (8.1) shows that the computed scheduling latency of entropy-based algorithm is comparatively lower than the EDF and RM as well. In order to execute tasks on or before deadline real time environment has been used. Therefore, the deadline-missing rate is computed in figure (8.2) in which we can see that entropy-based algorithm gives healthier results than existing algorithms. Moreover, figure (8.3) is explaining the migration rate that is used for balancing the load in-between processors. Entropy based algorithm gives better migration rate comparatively. Our last parameter is execution ratio that gives excellent results as compared to existing scheduling algorithms. Our explanation behind these results is that entropy decides the acceptance of task on the basis of space and time both.

8.4 *Summary*

In this chapter, independent real time tasks are generated in loosely coupled distributed system. In order to perform load balancing among the processors, scheduling and execution of tasks; we have replaced utilization factor with entropy factor. Further, simulation of existing algorithms EDF, RMS and entropy-based algorithm has been done on the same data. After the evaluation of resultant parameters we have got comparatively good results of the proposed algorithm. In future, this will be implementing on dependent tasks in which task duplication will be used for load balancing and reduction of execution time.

CHAPTER 9

CONCLUSION AND FUTURE WORK

9.1 *Summary*

The focus of this thesis remains on proposing and accepting a dynamics governing parameter that plays main role during generation, scheduling, execution, migration, and duplication of tasks. At the very beginning thesis deals with usual dependent tasks that means tasks having no deadline. In order to reduce the schedule length of directed acyclic graph (DAG) as well as balance the load among processors task duplication scheduling algorithm (*TDASLM*) came in to play. Afterwards, one common drawback of overloading that can occur due to task duplication has been discussed and then it is solved by task migration technique. In this way focus switch towards task migration method. Further author took turn towards real time tasks (tasks with deadline). Initially, independent tasks are simulated with new real time scheduling algorithm where task migration is allowed. For this, a *Joint EDF-RM scheduling algorithm* for real time task migration is discussed that resolves the problem of Domino's Effect of EDF scheduling algorithm.

Now, we draws attention to the backbone of real time distributed system *i.e.* utilization factor. So far, utilization is the only parameter that helps to govern the dynamics of any distributed system. Hence, the author has devised an alternative of utilization that behaves in parallel (one to one mapping) during arrival and execution of real time tasks. After getting one to one mapping, author puts forward to replace this new parameter entropy with utilization. This new parameter is information theoretic entropy that works by getting probabilities of the information of tasks (event) *i.e.* arrival time, deadline, period and worst-case execution time of the real time task. Finally, authors use this entropy parameter in place of utilization and applied it on earliest deadline first scheduling algorithm that works comparatively better than a normal EDF and RMS algorithm.

9.2 *Future Work*

Journey of entropy as a dynamics governing parameter has just begun with the EDF scheduling algorithm on independent tasks. Further, we will complicate it by implementation on real time DAG (Directed acyclic graph) with task duplication methodology. We shall attempt to determine the best-case scenario and direct applications. Testing the same for its scaling up capabilities as claimed briefly here, this remains another futuristic dimension for explorations. More advance model shall be rigorously investigated in the next scope of our approach. After that in future, this journey will move towards other scheduling algorithms and take a turn on the side of huge distributed or complex systems i.e. Grid computing. We are anticipating that meritoriously entropy can take over utilization parameter completely with certain additional advantages.

REFERENCES

[A. Bashir, 2013] A.Bashir, A. Madani Sajjad, Jawad Haider Kazmi, and Kalim Qureshi, Task Partitioning and Load Balancing Strategy for Matrix Applications on Distributed System, Journal of Computers 8, no. 3, 2013, pp. 576-584.

[A. Bestavros, 1996] A. Bestavros, Load Profiling in Distributed Real-Time Systems, 1996.

[A. Burchard, 1995] A.Burchard, J. Liebeherr, Y. Oh, and S. Son, New strategies for assigning real-time tasks to multiprocessor systems, IEEE transactions on computers, 1995, pp. 1429-1442.

[A. D. Ramírez, 2012] A.D.Ramírez, Dulce K. O., Pedro Mejía-Alvarez, A Multiprocessor Real-Time Scheduling Simulation Tool, 22nd International Conference on Electrical Communications and Computers (CONIELECOMP), 2012, pp.157-161.

[A. Gantman, 1998] A.Gantman, Pei-Ning Guo, James Lewis, and Fakhruddin Rashid, Scheduling real-time tasks in distributed systems: A survey, 1998.

[A. Mohammadi, 2005] A.Mohammadi and Selim G.Akl, Scheduling algorithms for Real-Time Systems, Technical Report No. 2005-499, University of Maryland at College Park, 2005.

[A. Sarkar, 2011] A.Sarkar,F.Mueller and H.Ramaprasad, Predictable task migration for locked caches in multi-core systems, In Proceedings of LCTES'11,2011,pp. 131-140.

[A. Srinivasan, 2003] A.Srinivasan, J.H. Anderson, Efficient scheduling of soft real-time applications on multiprocessors, In Proceedings of the 15th Euromicro Conference on Real-Time Systems, 2003, pp.51–59.

[A. Tanenbaum, 2002] A.Tanenbaum and M. Van Steen, Distributed Systems: Principles and Paradigms,Prentice Hall, Pearson Education, USA, 2002.

[AoPS Incorporated and Solving, 2006] Articles and Excerpts, AoPS Incorporated and Solving, A.P., vol.1, ISBN 9781934124000, http://books.google.co.in/books?id=x2YxTUc3G5kC, United States of America, 2006.

[B. Kruatrachue, 1988] B.Kruatrachue and T. Lewis, Grain Size Determination for Parallel Processing, IEEE Softw., 1988, pp. 23-32.

[B. T. Akgün, 1996] B.T. Akgün, et al., An Implementation of a Load Balancing Algorithm for the BAG System, 14th IASTED International Conference on Applied Informatics, Austria, 1996, pp.43-45.

[B. T. Akgün, 2001] B.T. Akgün, BAG Distributed Real-Time Operating System and Task Migration, Turkish Journal Electrical Engineering, Vol. 9, 2001, pp.123-136.

[C. E. Shannon, 1949] C.E. Shannon, A Mathematical Theory of Communication, Bell Systems Technical Journal, 1949, pp.1-54.

[C. L. Liu, 1973] C.L.Liu and James W. Layland, Scheduling algorithms for multi-programming in a hard-real-time environment, Journal of the As-sociation for Computing Machinery, Vol.20(1), 1973, pp. 46–61.

[C. Lu, 2004] C.Lu, X. Wang, and X. Koutsoukos, End-to-End utilization control in distributed real-time systems, International Conference on Distributed Computing Systems (ICDCS), Tokyo, Japan, 2004.

[C. Wang, 2007] C.Wang and W.R. Dieter, Power-Aware Task Assignment for Priority-Driven Distributed Real-Time System, Technical Report ECE–2007–10–17, University of Kentucky, 2007.

[D. Bozdag, 2006] D.Bozdag, Umit Catalyurek and Fu¨sun O¨ zgu¨ner, A Task Duplication Based Bottom-up scheduling algorithm for heterogeneous environments, 20th International Parallel and Distributed Processing Symposium, 2006, pp. 1-12.

[D. Chen, 1998] D.Chen, A. Mok, S. Baruah, On Modeling Real-Time Task Systems, Proceedings of The European Educational Forum School on Embedded Systems, in LNCS, Springer Verlag No. 1494, 1998, pp.153-169.

[D. Feldman, 2002] D. Feldman, A Brief Introduction to: Information Theory, Excess Entropy and Computational Mechanics, October 2002.

[D. Ferrai, 1987] D.Ferrai, Zhou, S., An Empirical Investigation of load indices for load balancing applications, 1987.

[D. R. Cheriton, 1988] D.R.Cheriton, The V Distributed System, Comm. of the ACM, Vol. 31(3) 1988, pp.314-333.

[D. Sekhar, 1997] D. Sekhar and D.P.Agrawal, A task duplication based scalable scheduling algorithm for distributed memory systems, Journal of parallel and Distributed Computing Vol. 46(1), 1997, pp.15-27.

[Dong Yu, 2009] Dong Yu, Li Deng, Alex Acero, Using continuous features in the maximum entropy model, Pattern Recognition Letters 30, 2009, pp.1295–1300.

[E. G. Coffman, 1998] E.G.Coffman, G. Galambos, S. Martello and D. Vigo, Bin Packing Approximation Algorithms: Combinational Analysis, Kluwer Academic Publishers, Ed. D. Z. Du and P.M. Pardalos, 1998.

[E. T. Jaynes, 1957] E.T.Jaynes, Information Theory and Statistical Mechanics, Physical Review, vol. 106 (4), 1957, pp.620-630.

[G. Buttazzo, 1995] G.Buttazzo, M Spuri and F.Sensini,Value and Deadline scheduling in Overload Conditions,Real Time systems, ,1995, pp.90-99.

[G. C. Buttazzo , 2003] G.C.Buttazzo, Rate monotonic vs. EDF: Judgment day, In Proceedings of the 3rd International Conference on Embedded Software, Philadelphia, 2003, pp.67-83.

[G. Couloris, 2001] G.Couloris, J.Dollimore, and T. Kinberg, Distributed Systems – Concepts and Design, 4th Edition, Addison-Wesley, Pearson Education, UK, 2001.

[G. Umarani, 2012] G.Umarani Srikanth, A. P. Shanthi, V. Uma Maheswari et. al., A Survey on Real Time Task Scheduling, European Journal of Scientific Research, Vol.69(1), 2012, pp.33-41.

[H. C. Wang, 2011] H.C.Wang and C.W.Yao, Task Migration for Energy Conservation in Real-TimeMulti-processor Embedded Systems, International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 2011, pp 393-398.

[J. A. Stankovic, 1988] J.A.Stankovic and K. Ramamritham, Tutorial Hard Real-Time Systems, IEEE Computer Society Press, 1988.

[J. Anderson, 2005] J.Anderson, V. Bud and U. C. Devi, An EDF-based scheduling algorithm for multiprocessor soft real-time systems, 17th IEEE Euromicro Conference on Real Time Systems, 2005, pp. 199-208.

[J. Anderson, 2008] J.Anderson, V. Bud and U. C. Devi, An EDF-based Restricted-Migration Scheduling Algorithm for Multiprocessor Soft Real-Time Systems, Real-Time Systems , Vol.38(2), 2008, pp. 85-131.

[J. Baxter, 1989] J.Baxter and J.H. Patel, The LAST Algorithm: A Heuristic-Based Static Task Allocation Algorithm, International Conference on Parallel Processing, 1989, pp. 217-222.

[J. Carpenter, 2004] J.Carpenter, S.Funk, P.Holman, A.Srinivasan et. al., A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms, in Handbook of Scheduling: Algorithms, Models and Performance Analysis, Florida, 2004, pp. 30-1 - 30-19.

[J. E. G. Coffman, 1996] J.E.G. Coffman, M.R. Garey and D.S. Johnson, Approximation algorithms for bin packing: a survey, Approximation algorithms for NP-hard problems, 1996, pp. 46-93.

[J. Goossens, 1999] Joel Goossens, Scheduling of Hard Real-Time Periodic Systems with Various Kinds of Deadline and Offset Constraints, Ph.D Thesis, Universite Libre De Bruxelles, Belgium, 1999.

[J. Goossens, 2002] Joel Goossens, S. Baruah and S. Funk, Real-time scheduling on multiprocessor, 10th International Conference on Real-Time System, 2002.

[J. Lehoczky, 1989] J. Lehoczky, Lui Sha and Ye Ding, The Rate Monotonic Scheduling Algorithm: Exact characterization and Average case behavior, IEEE Real Time Systems Symposium, 1989, pp.166-171.

[J. Lopez, 2000] J.Lopez, J.Diaz, M.Garcia and D. Garcia, Worst-case utilization bound for edf scheduling on real-time multiprocessor systems, In Proceedings of the 12th Euromicro Workshop on Real-Time Systems, 2000, pp. 25-33.

[J. Singh, 2011] J.Singh and H.Singh, Efficient Task Scheduling for heterogrnous multiprocessor using Genetic Algorithm with Node Duplication, Indian Journal of Computer Science and Engineering, 2(3), 2011, pp. 402-410.

[J. Singh, 2012] Jagbir Singh and Satyendra Prasad Singh, Schedulability Test for Soft Real-Time Systems under Multiprocessor Environment by using an Earliest Deadline First Scheduling Algorithm, arXiv preprint arXiv:1205.0124, 2012.

[J. W. S. Liu, 2000] J.W.S. Liu, Real-Time Systems, Prentice Hall, Upper Saddle River, NJ, 2000.

[J. W. S. Liu, 2003] J.W.S.Liu, Real-Time Systems, Pearson Education, Delhi, India, 2003.

[Jiaying Zhang , 2006] Jiaying Zhang and Peter Honeyman, Naming, Migration, and Replication for NFSv4, 5th International Conference on System Administration and Network Engineering, Ann Arbor 1001, 48103-4978, January 2006.

[K. Kotecha, 2010] K.Kotecha and A.Shah, Efficient Scheduling Algorithms for Real-Time Distributed Systems, In Proceedings of the 1st International Conference on parallel, Distributed and Grid Computing,2010, pp 44–48.

[K. Nadiminti, 2006] K.Nadiminti, Marcos Dias de Assunção and R. Buyya, Distributed Systems and Recent Innovations: Challenges and Benefits, InfoNet Magazine, Vol.16, 2006, pp.1–5.

[K. Ramamritham, 1990] K. Ramamritham, John A. Stankovic and P-F. Shiah, Efficient scheduling algorithms for real-time multiprocessor systems, IEEE Transactions on Parallel and Distributed Systems, Vol.1(2), 1990, pp.184-194.

[L. Brillouin, 2004] L.Brillouin, Science & Information Theory, Dover Publications, 2004, pp. 293.

[L. Kleinlock, 1976] L.Kleinrock, Queueing Systems – Vol.2: Computer Applications. Wiley Interscience, 1976.

[L. Kleinlock, 1985] L.Kleinlock, Distributed Systems, Communications of the ACM,1985.

[L. M. Ni, 1985] L.M.Ni and K.Hwang, Optimal load balancing in a multiple processor system with many job classes, IEEE Transantion on Software Eng., Vol.SE-11(5), 1985, pp. 491-496.

[L. N. Bhuyan, 1984] L.N.Bhuyan and D.P. Agrawal, Generalized Hypercube and Hyperbus structures for a Computer Network, IEEE Transactions on computers, Vol.C-33(4), 1984, pp. 323-333.

[Lo. V. M., 1988] Lo.V.M., Heuristic Algorithms for Task Assignment in Distributed Systems, IEEE Transactions on computers, Vol.37(11), 1988, pp. 1384-1397.

[M. Bertogna, 2009] M.Bertogna, Cirinei, M.,Lipari,G., Schedulability Analysis of Global Scheduling Algorithms on Multiprocessor Platforms, IEEE Transactions on Parallel and Distributed Systems, Vol.20(4), 2009, pp.553-566.

[M. Joseph, 1996] M. Joseph, Real-time Systems: Specification, Verification and Analysis, Prentice Hall, 1996.

[M. Mitchell, 2001] M.Mitchell, Use Of Directed Acyclic Graph Analysis in Generating Instructions for Multiple Users, Asia-Pacific symposium on Information visualisation, Australian Computer Society, Vol.9,2001.

[N. Fisher, 2006] Nathan Fisher, Sanjoy Baruah et. al., The partitioned scheduling of sporadic tasks according to static-priorities, 18th IEEE Euromicro Conference on Real-Time Systems, 2006.

[N. W. Fisher, 2007] N.W. Fisher, The Multiprocessor Real-Time Scheduling of General Task Systems, Ph.D. dissertation. Department of Computer Science, The University of North Carolina at Chapel Hill, NC, 2007.

[O. Zapata, 2005] O. Zapata, U. Pereira, and Pedro Mejıa Alvarez, Edf and RM multiprocessor scheduling algorithms: Survey and performance evaluation., Seccion de Computacion Av. IPN 2508, 2005.

[O'Leary, 2003] O'Leary, The Structure of Proof: With Logic and Set Theory, Prentice-Hall 2003.

[P. A. Laplante, 1993] P.A.Laplante, Real-time Systems Design and Analysis, An Engineer Handbook, IEEE Computer Society , 1993.

[P. Chaudhuri, 2010] P.Chaudhuri and J. Elcock, Process Scheduling in Multiprocessor Systems Using Task Duplication, International Journal of Business Data Communications and Networking, Vol.6(1), 2010, pp. 58-69.

[P. Emberson, 2007] P.Emberson, Iain Bate, Minimizing Task Migration And Priority Changes In Mode Transitions, 13th IEEE Real Time and Embedded Technology and Applications Symposium(RTAS'07), 2007, pp.158-167.

[P. Li, 2004] P.Li and B.Ravindran, Fast, Best-Effort Real-Time Scheduling Algorithms, IEEE Transactions on Computers, Vol.53(9), 2004, pp. 1159-1175.

[P. Penfield, 2003] P.Penfield, Chapter-10, Principle of Maximum Entropy, Version 1.0.2, www.mtl.mit.edu/Courses/6.050/2003/notes/chapter10.pdf, 2003.

[R. Malouf, 2002] R.Malouf, A comparison of algorithms for maximum entropy parameter estimation, In Proceedings of the Sixth Conference on Natural Language Learning (CoNLL), 2002, pp. 49–55.

[R. Sharma, 2011] Rashmi Sharma and Nitin, Duplication with Task Assignment in Mesh Distributed System, IEEE World Congress on Information and Communication Technologies (WICT), 2011, pp. 672-676.

[R. Sharma, 2012] Rashmi Sharma and Nitin, Optimal Method for Migration of Tasks with Duplication, 14th International Conference on Modelling and Simulation, 2012, pp. 510-515.

[R. Sharma, 2012] Rashmi Sharma, Nitin, Task Migration with EDF-RM Scheduling Algorithms in Distributed System, International Conference on Advances in Computing and Communications, 2012, pp. 182-185.

[R. Sharma, 2013] Rashmi Sharma, Nitin, Visualization of Information Theoretic Maximum Entropy Model in Real Time Distributed System, In proceedings of International Conference on Advances in Computing and Communications, 2013, pp.282-286.

[R. Sharma, 2014] Rashmi Sharma and Nitin, Performance Evaluation of New Joint EDF-RM Scheduling Algorithm for Real Time Distributed System, Journal of Engineering, 2014.

[S. Dhakal, 2007] S.Dhakal, M. Majeed Hayat and E. Jorge Pezoa, Dynamic Load Balancing in Distributed Systems in the Presence of Delays: A Regeneration-Theory Approach, IEEE Transactions on Parallel and Distributed Systems, Vol.18 (4), 2007, pp.485-497.

[S. F. Gull, 1984] S.F. Gull and J. Skilling, Maximum entropy method in image processing, IEEE Proceedings Vol. 131(6), 1984, pp.646-659.

[S. Ranaweera, 2000] S.Ranaweera and D.P. Agrawal, A Scalable Task Duplication Based Scheduling Algorithm for Heterogeneous Systems, International conference on Parallel Processing, 2000, pp. 383.

[S. Ranaweera, 2002] S.Ranaweera and D.P.Agrawal, A Task Duplication Based Scheduling Algorithm For Heterogenous System, 14th International Parallel and Distributed Processing Symposium, 2002, pp.445-450.

[S. Shimokawa, 2001] S.Shimokawa, H. Ozawa, On the thermodynamics of the oceanic general circulation: entropy increase rate of an open dissipative system and its surroundings, Tellus, 2001, pp.266–277.

[Shu-Kun Lin, 1999] Shu-Kun Lin, Diversity and Entropy, Entropy, 1999, pp.1–3.

[T. P. Baker, 2005] T.P.Baker, A comparison of Global and Partitioned EDF Schedulability Tests for Multiprocessor, TR-051101, 2005.

[T. T. Y. Suen, 1992] T.T.Y. Suen, J.S.K.Wong, Efficient Task Migration Algorithm for Distributed Systems, IEEE Transactions on Parallel and Distributed Systems, Vol.3(4), 1992, pp. 488-499.

[V. Darera, 2006] V.Darera, 2006 and J. Lawrence, Utilization bounds for RM scheduling on uniform multiprocessors, 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2006.

[W. H Zurek, 1989a] W.H Zurek, 1989a Algorithmic randomness and physical entropy, Physical Review A, 40, 1989a, pp. 4731–4751.

[W. H Zurek, 1989b] W.H Zurek, Thermodynamic cost of computation, algorithmic complexity and the information metric, Nature, 347, 1989b, pp. 119–124.

[W. R. Derek, 2008] W. R. Derek, Entropy and Uncertainty, Entropy 10, 2008, pp. 493-506.

[X. Wang, 2005] X.Wang, Jia, D., Lu, C., Koutsoukos, X., Decentralized Utilization Control in Distributed Real-Time Systems, 26th IEEE International Real-Time Systems Symposium, 2005.

[X. Wang, 2007] X.Wang, Jia, D., Lu, C., Koutsoukos, X., DEUCON: Decentralized End-to-End Utilization Control for Distributed Real-Time Systems, IEEE Transactions on Parallel and Distributed Systems, Vol. 18(7), 2007, pp.996-1009.

[Y. J. Singh, 2009] Y.J. Singh, S. C. Mehrotra, An Analysis of Real Time Distributed System under different priority policies, World Academy of Science, Engineering and Technology, 2009.

[Y. Jegou, 1997] Yvon Jégou, Runtime Support for Task Migration on Distributed Memory Architectures, 11th International Parallel Processing Symposium, IPPS'97, 1997.

LIST OF PUBLICATIONS

International Journals Published:

[1]. Rashmi Sharma and Nitin, Duplication with task assignment in Mesh Distributed System, Journal of Information Processing Systems (JIPS), Vol.10, No.2, pp.193-214, June 2014.

[2]. Rashmi Sharma and Nitin, Performance Evaluation of New Joint EDF-RM Scheduling Algorithm for Real Time Distributed System, Journal of Engineering, Hindawi publication Corporation, Volume 2014 (2014), Article ID 485361, 13 pages, January 2014.

[3]. Rashmi Sharma and Nitin, Entropy, a New Dynamics Governing Parameter in Real Time Distributed System: A Simulation Study, International Journal of Parallel, Emergent and Distributed Systems, Taylor and Francis, Vol. 29, No. 6, pp. 562-586,12 November 2013.

International Conference Papers Published:

[1]. Rashmi Sharma and Nitin, Evaluation and Comparison of Load Balancing in RTDS using Information Theoretic Entropy, Proceedings of 4th IEEE International Advance Computing Conference (IEEE IACC), ITM University, INDIA, pp. 674-679, February 21-22, 2014.

[2]. Rashmi Sharma and Nitin, Visualization of Information Theoretic Maximum Entropy Model in Real Time Distributed System, Proceedings of the 3rd IEEE International Conference on Advances in Computing and Communications (IEEE ICACC), Kerala, INDIA, August 29-31, 2013, pp. 282-286.

[3]. Rashmi Sharma and Nitin, Task Migration with EDF-RM Scheduling Algorithms in Distributed System, Proceedings of the 2nd IEEE International Conference on Advances in Computing and Communications, Kerala (IEEE ICACC), INDIA, August 9-11, 2012, pp. 182-185.

[4]. Rashmi Sharma and Nitin, Optimal Method for Migration of Tasks with Duplication, Proceedings of the 14th IEEE International conference on Computer Modeling and Simulation (IEEE UKSIM), Emmanuel College, Cambridge, UK, March 28-30, 2012, pp. 510-515.

[5]. Rashmi Sharma and Nitin, Duplication with Task Assignment in Mesh Distributed System Scheduling, Proceedings of the IEEE World Congress on Information and Communication Technologies, University of Mumbai, INDIA, December 11-14, 2011, pp. 672-676.