

0.2
c.i.
10

**PERFORMANCE EVALUATION OF DYNAMIC LOAD
BALANCING ALGORITHMS IN DISTRIBUTED COMPUTING
SYSTEMS**

By

Rania Ali Najy Etoom

Supervisor

Dr. Sami I. Serhan

**This Thesis was Submitted in Partial Fulfillment of the Requirements for the
Master's Degree of Computer Science.**

Faculty of Graduate Studies

The University of Jordan

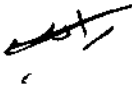
تعتمد كلية الدراسات العليا
هذه النسخة من الرسالة
التوقيع التاريخ 19/8/2010

August, 2010

The University of Jordan

Authorization Form

I, Rania Ali Naji Etoom, authorize the University of Jordan to supply copies of my Thesis/ Dissertation to libraries or establishments or individuals on request, according to the University of Jordan regulations.

Signature: 

Date: 12/8/2010.

COMMITTEE DECISION

This Thesis /Dissertation (Performance Evaluation of Dynamic Load Balancing Algorithms in Distributed Computing Systems) was Successfully Defended and Approved on 28/7/2010.

Examination Committee

Signature

Dr.Sami I. Serhan, (Supervisor)



Associate Professor of Computer Design

Dr.Mohammad Sulieman Qataweh, (Member)



Associate Professor of Parallel Systems and Networks

Dr.Basel Ali Mahafzah, (Member)



Assistant Professor of Distributed & Parallel Computing, and Interconnection Networks.

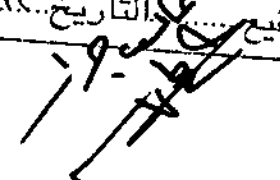
Dr. ZIAD A.A. ALQADI, (Member)



Associate Prof. Of computer engineering

(Al-Balqa `Applied University)

تعتمد كلية الدراسات العليا
هذه النسخة من الرسالة
التوقيع التاريخ ٢٨/٧/٢٠١٠



DEDICATION

This work dedicated to my father, mother, brothers, sisters, and husband.

AKNOWLEDGEMENT

I would like to express my deepest appreciation to Dr. Sami I. Serhan for his support to finish this work.

TABLE OF CONTENTS

Subject	Page
Committee Decision	II
Dedication	III
Acknowledgement	IV
List of Contents.....	V
List of Tables	VII
List of Figures and Plates.....	IX
List of Equations.....	XI
List of Abbreviations	XII
Abstract	XV
1. Introduction	1
1.1. Distributed Systems.....	1
1.2. Load Balancing.....	2
1.2.1. Load Balancing Approaches.....	3
1.2.2. Load Balancing Categories.....	5
1.2.3. Load Balancing Policies.....	7
1.2.4. Load Balancing Parameters.....	8
1.3. Thesis Organization.....	10
2. Background and Literature Review.....	11
2.1 Introduction.....	11
2.2 Related Work.....	20
3. Enhanced Dynamic Biasing Algorithm for Load Balancing in Hierarchical	

Distributed Systems.....	23
3.1. Overview.....	23
3.2. The System Model.....	24
3.3. The Proposed Scheme Assumptions.....	26
3.4. Determining Suitable Time Intervals.....	27
3.5. Enhanced Dynamic Biasing Algorithm with CPU Utilization (EDBA-CPU-U)..	28
3.5.1. Methodology.....	28
3.6. Enhanced Dynamic Biasing Algorithm with Process Migration (EDBA-PM)..	30
3.6.1. Methodology.....	31
3.6.2 Process Migration Policy Used in the Enhanced Scheme.....	32
3.7. Simulation Cases.....	34
4. Results and Evaluation.....	35
4.1. Introduction.....	35
4.2 The Simulator.....	35
4.3. Scenarios of the Simulation	38
4.4. Performance Metrics.....	39
4.5. Results and Discussions	39
4.5.1. Homogeneous Distributed Systems.....	40
4.5.1.1 Constant Arrival Rate of Tasks.....	40
4.5.1.1.1 EDBA-CPU-U.....	40
4.5.1.1.2 EDBA-PM.....	43
4.5.1.2 Variable Arrival Rate of Tasks.....	44

4.5.1.2.1 EDDBA-CPU-U.....	45
4.5.1.2.2 EDDBA-PM.....	46
4.5.2. Heterogeneous Distributed Systems.....	47
4.5.2.1. Constant Arrival Rate of Tasks.....	48
4.5.2.1.1 EDDBA-CPU-U.....	48
4.5.2.1.2 EDDBA-PM.....	49
4.5.2.2. Variable Arrival Rate of Tasks.....	51
4.5.2.2.1 EDDBA-CPU-U.....	51
4.5.2.2.2 EDDBA-PM.....	53
4.6 Results Summary.....	54
5. Conclusions and Future Works.....	55
5.1. Conclusions.....	55
5.2 Future Works.....	56
References	57
Abstract (in the second language).....	65

LIST OF TABLES

NUMBER	TABLE CAPTION	PAGE
1.1	Categories of Load Balancing Techniques	6
4.1	Determining Simulation Intervals	35

LIST OF FIGURES

NUMBER	FIGURE CAPTION	PAGE
1.1	Distributed Computing Systems Types	1
2.1	Underlying Distributed System	21
3.1	Underlying Distributed System Structure	25
3.2	Simulation cases	33
4.1	The Node level of EDDBA-CPU-U Algorithm.	36
4.2	The Process Model of LLB	37
4.3	The code of the arrival state of the FSM	37
4.4	Response time with varying number of tasks for EDDBA-CPU-U with constant arrival rate of tasks in homogeneous DS	40
4.5	Response time with varying number of processors for EDDBA-CPU-U with constant arrival rate of tasks in homogeneous DS	41
4.6	Throughput with varying number of tasks for EDDBA-PM with constant arrival rate of tasks in homogeneous DS	42
4.7	Throughput with varying number of processors for EDDBA-PM with constant arrival rate of tasks in homogeneous DSs	43
4.8	Response time with varying number of tasks for EDDBA-CPU-U with variable arrival rate of tasks in homogeneous DS	44
4.9	Response time with varying number of processors for EDDBA-CPU-U with variable arrival rate of tasks in homogeneous DS	45

4.10	Throughput with varying number of tasks for EDDBA-PM with variable arrival rate of tasks in homogeneous DS	45
4.11	Throughput with varying number of processors for EDDBA-PM with variable arrival rate of tasks in homogeneous DS	46
4.12	Response time with varying number of tasks for EDDBA-CPU-U with constant arrival rate of tasks in heterogeneous DS	47
4.13	Response time with varying number of processors for EDDBA-CPU-U with constant arrival rate of tasks in heterogeneous DS	48
4.14	Throughput with varying number of tasks for EDDBA-PM with constant arrival rate of tasks in heterogeneous DS	49
4.15	Throughput with varying number of processors for EDDBA-PM with constant arrival rate of tasks in heterogeneous DS	49
4.16	Response time with varying number of tasks for EDDBA-CPU-U with variable arrival rate of tasks in heterogeneous DS	50
4.17	Response time with varying number of processors for EDDBA-CPU-U with variable arrival rate of tasks in heterogeneous DS	51
4.18	Throughput with varying number of tasks for EDDBA-PM with variable arrival rate of tasks in heterogeneous DS	52
4.19	Throughput with varying number of processors for EDDBA-PM with constant variable rate of tasks in heterogeneous DS	52

LIST OF EQUATIONS

NUMBER	EQUATION CAPTION	PAGE
1	Load Evaluation	28
2	State Information exchange policy	28
3	Biases Calculation	29
4	Load Evaluation	30
5	Decision of process migration	30
6	State Information exchange policy	30
7	Biases Calculation	31
8	Percentage Difference	36

LIST OF ABBREVIATIONS

ATS	The Augmented Tabu-Search Algorithm
CORBA	Common Object Request Broker Architecture
CPU-U	CPU Utilization
DAG	Direct Acyclic Graph
DBA	Dynamic Biasing Algorithm
DCS	Distributed Computing System
DLB	Dynamic Load Balancing
DM	Decision Making
DNS	Domain Name System
DOS	Distributed Operating System
DR	Designated Representative
DS	Distributed System
EDBA	Enhanced Dynamic Biasing Algorithm
EDBA-CPU-U	Enhanced Dynamic Biasing Algorithm-CPU-Utilization
EDBA-PM	Enhanced Dynamic Biasing Algorithm-Process Migration.
EPSTS	Enhanced Positional Scan Task Scheduling Algorithm
FIFO	First-In-First-Out
GLB	Global Load Balancer
HEFT	Heterogeneous Earliest-Finish Time Algorithm
HPF	High Performance Fortran
HPTS	Heterogeneous Parallel Task Scheduler

HL	Heavily Loaded
ISR	Interrupt Service Routine
IWM	Incremental Weight migration
JI	Job Interval
JQ	Job Quantity
LB	Load Balancing
LIB	Load Imbalance
LL	Lightly Loaded
LLB	Local Load Balancer
MAS	Multi-Agent System
ML	Moderately Loaded
MLSRR	Minimum Load State Round Robin
ORB	Object Request Broker
OQ	Ordinary Queue
OS	Operating System
PN	Primary Node
PS	Processing Speed
PSI	Periodic Symmetrically-Initiated
PCLS	Positional Scan Load Balancing algorithm
PSTS	Positional Scan Task Scheduling algorithm
RNS	The Recursive Neighbor Search algorithm
RQ	Restart Queue
SLB	Static Load Balancing

TI	Transfer Interval
TP	Transfer Policy
TS	Transfer Size
TT	Transfer Threshold
SN	Supporting Node
TA	Task Assignment
TAPTF	Task Assignment based on Prioritizing Traffic Flows
TAPTF-WC	Task Assignment based on Prioritizing Traffic Flows with Work-Conserving Migration

**PERFORMANCE EVALUATION OF DYNAMIC LOAD
BALANCING ALGORITHMS IN DISTRIBUTED COMPUTING
SYSTEMS**

BY

Rania Ali Najy Etoom

Supervisor

Dr. Sami I. Serhan

In this study we focus on achieving load balancing in distributed systems of hierarchical structure by enhancing a dynamic algorithm. In the Dynamic Biasing Algorithm (DBA) the current load state is measured by CPU queue length to distribute load among nodes based on their current load state.

Two proposed algorithms are presented, in the first algorithm the load state is measured by the CPU queue length as in the original algorithm DBA, in addition to the CPU utilization rate. This new load index (CPU utilization) increases the system performance because it makes the nodes busy most of the time. It increases the system performance by 62.9% in terms of response time factor.

The second proposed algorithm is done by inserting the job migration policy among nodes. A specific threshold that is changed adaptively is compared by the load state of nodes periodically, if the node is heavily loaded (its load state is larger than threshold),

then the process migration policy is executed by transferring some of its load to another node.

This enhancement increases the performance of the system in terms of throughput especially when the system is heavily loaded. The migration policy helps in achieving the load balancing because it makes all nodes have the same load approximately, and increasing the system performance accordingly. It increases the system performance in terms of throughput factor by 37%.

1. Introduction

The fast expansion of utilizing computers including software and hardware, increases the need for resource sharing applications, and increases the workload across the Internet. On the other hand, it eases the process of linking various distributed computers through the network, allowing them to execute the load in parallel and to cooperate in finishing some tasks to decrease the turnaround time.

Most of the cases the load is too much and becomes the bottleneck of the system, this problem can be solved by increasing the size of the computers in the Distributed System, or distribute the load among the processing nodes in an efficient manner. The load redistribution termed Load Balancing.

1.1. Distributed Systems

Distributed Computing System (DCS) is a set of processing nodes that can be PC's, workstations, etc. interconnected together by a network, they cooperate together to finish a specific task in a minimum amount of time. DCS has the advantage of resource sharing among the nodes eased by the network, which increases the performance and the reliability of the system. Figure 1.1 shows the types of DCS.

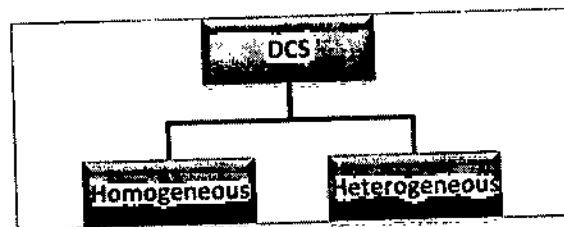


Figure 1.1: Distributed Computing Systems Types

Heterogeneity can be in networks, computer hardware, operating systems, and programming languages, Heterogeneous DCS is constructed from several processing nodes that differ in the previous characteristics. And all nodes in a Homogeneous DCS have the same properties.

There are two main reasons for using distributed systems and distributed computing:

- 1) Type of the application may need the use of a communication network that connects several computers.
- 2) It is more beneficial to use a cluster of several low-end computers instead of using one high-end computer because, we can get the desired level of performance in a less cost and the DCS is easier to expand than uni-processor system, moreover the DCS is more reliable than the other non-DCS, as there is no single point of failure.

1.2. Load Balancing

Chen, et al. (2008), Devine, et al. (2005), Spies (1996), and Sharma, and et al. (2008), contribute that each node in the DCS is assigned a load, which is the amount of the computational tasks that are taking the service or waiting for a service, and hence, Workload of a node is the amount of time needed to accomplish all tasks assigned to the node. The spatiotemporal assignment is described by two parameters:

- 1) JI (Job Interval): the time interval between two different load assignment.
- 2) JQ (Job Quantity): the size of tasks to be accomplished by each assignment.

Each node in the DCS has a two types of jobs, the Dedicated and the Generic jobs; it has variants of the Dedicated jobs that impose constraints on the average response time, and these jobs must be executed locally (in the node that generates it), in the other hand; Generic tasks can be executed at any node in the system (ROSS, and YAO, 1991).

Consider a university, which forms a DCS and comprised of many departments or faculties. Each department has its own computer system and resources needed to execute and process its tasks. When a specific department generates a task, then this task has two cases of execution; first it must be processed locally in the same

department that generates it because, it requires its own resources. Or it is commonly the case that this job is generic and can be processed in any computer system in the other departments. Moreover, a given computer system in any department can have a lot of jobs to execute, so it is Heavily-Loaded (HL), whereas another system is idle or Lightly-Loaded (LL). Then, in order to improve the system performance, specifically the average response time of jobs, an efficient Load Balancing (LB) mechanism should be used to transfer some Generic jobs from the HL departments to the LL or the idle departments, and distribute the load among the departments in a judicious way. Hence we need two types of scheduling algorithms to control the assignment of jobs to nodes:

- ❖ Local scheduling: it is found in each node in the OS level, by distributing the tasks to the time slots of the CPU.
- ❖ Global scheduling: it is the distribution of tasks among nodes in the DCS, represented by LB, which is process of redistribution the generic jobs among all nodes to improve the performance of the Distributed System (DS). It ensures that each node in the DS performs the same number of tasks approximately at any moment of time, and if the load is balanced in the system, then you will not find a HL- node and an idle node or a LL node at the same time.

LB algorithms are used to minimize the overhead of the communication network, increasing the efficiency of the system in term of utilization rate, and the average response time of the all tasks in the system, and Achieving the fairness among the competitive applications, (Razzaque, and Seon Hong, 2006), and (Sharma, et al. 2008).

1.2.1 Load Balancing Approaches

Kwok, and Cheung, (2004), and Choi (2004); LB is needed to distribute the tasks among the processing nodes in the DS in an effective way, to improve the performance

of the system, and minimize the average response time, by using one of the following approaches:

1. Network (application software) level: it is found in the Distributed-Web architecture, and it is based on the Domain Name System (DNS) to apply the LB techniques. When a client needs a specific service, he will send a request that contains the symbolic name (the host name of the URL) to the DNS, and then the DNS will make the decision of load balancing and translate this symbolic name to an IP address of a particular server, to serve the request using the mapping table stored in the DNS. DNS can assign different servers with different IP addresses to the same request by using different policies such as the Round Robin (RR) or the Random policy, so the service needed by the client can be shared by more than one server.
2. Dispatcher based approach: it is found in the Distributed-Web Server network, in this approach one of the processing nodes in the DS (called the dispatcher), acts as a representative of all the other nodes in the system, this node has an IP address that is known to all other nodes (global). The dispatcher receives the requests from all clients' nodes in the DCS, and then it will make a LB decision to distribute the different requests to possibly different nodes in the system, the dispatcher commonly uses the simple policies, such as the RR or the Random policy.
3. Operating System (OS) approach (Software) level: in this approach the Distributed Operating System (DOS) provides the LB methods such as process migration¹, scheduling, etc, at the kernel level. For example, the process in the

¹ Process migration: the process of transferring tasks from the source node to the destination node during execution.

AMOEBA-DOS, consults the kernel to assign it to the CPU that is most LL. But not all kernels support such facilities, such as the Linux and the Windows desktops OSs.

4. Middleware level approach: middleware is a class of software that is designed to manage the heterogeneity and the complexity in the DSs. It is designed as a middle software layer between the OS and the application program that provides a common program abstraction across the DS; it is above the OS and below the distributed application.

The Common Object Request Broker Architecture (CORBA), which is a standard for distributed object computing is an example of the middleware that can perform the common network programming tasks such as Marshalling/Demarshalling, Error detection, and recovery.

When middleware-based LB is used in CORBA, it uses the Object Request Brokers (ORBs) to receive all clients' requests, and then distributes them to Distributed Objects in different remote servers depending on the LB decision based on LB algorithms, commonly it uses the Random, or the RR policies, that are incorporated in the ORBs.

1.2.2. Load Balancing Categories

Kwok, and Cheung, (2004), Chen, et al. (2008), Grosu, and Chronopoulos, (2005). Yang, et al. (2009) and Yan, et al. (2007) LB techniques may be divided into many categories, each of them is used depending on the type of applications used in the system, and we will give a brief description of these categories in Table.1.1

One of these categories is that LB technique can be static or dynamic. In this thesis we will focus on the dynamic approaches, which are the more complicated but has a better performance.

Table 1.1: Categories of Load Balancing Techniques

Division	First Category	Second Category
Location Of Decision Making (DM)	Centralized : There is a node in the DCS that has information about the entire system, and responsible for controlling the distribution of the loads to the other nodes in a balanced manner. The central node must be with a high Processing power and high communication speed, otherwise it will become the bottleneck of the system. It has a better performance. But if the central node fails functioning, then the LB is lost	Decentralized (Distributed): Each node in the DCS is responsible for information collecting and DM (Decision Making) independently. So, the DM is divided among all nodes in the network, and each one of them has its information about the entire system and responsible for its share of DM of scheduling.
Time Of Decision Making	Periodic: In every time interval, a DM and information collecting must be taken. The interval is constant and is determined in advance.	Non-periodic: The interval time is not constant. A DM and collecting information can be taken in any time, when the system needs that.
Changes to DM	Static: All decisions of controlling the LB are fixed and will not change as time goes, because the work load is distributed among the nodes in the beginning of execution depending on the processors performance. It has the disadvantage that allocation of a node to execute a specific task is determined when the task is created, and cannot be changed while task execution, or when the system conditions changed. Examples of the static policies are the RR and Random policy.	Dynamic: It takes the current load into consideration to make decision of LB more intelligently, because it distributes the work load at run time. It is more sophisticated than static but it has a better performance, because It uses the up-to-date state information of the system such as the load ratio of each node. It adds more overhead to the system, because it has to collect, store, and analyze state information of the system but it can cope with the dynamically changing circumstances of the system.

1.2.3 Load Balancing Policies

When a specific LB algorithm is applied, then it is conditioned with some LB policies, these policies are listed below:

- 1) Location Policy: it determines the destination node that will take part in accomplishing the load that cannot be finished soon by a busy node, its choices of choosing the destination node(s) are:
 - ❖ The destination node that will receive the load when it is transferred out from the busy node is one of its neighboring nodes, it is chosen according to its Processing Speed (PS), and utilization rate.
 - ❖ All neighboring nodes are selected to share the load at a specific ratio.
- 2) Transfer Policy (TP): It determines if the node can take part in transferring a task (as a sender or receiver), this can be done by determining if the node is busy (sender), or an idle node (receiver). It is determined by the three parameters listed below:
 - ❖ Transfer Threshold (TT): When to consider the node whether it is overloaded or not? This can be decided by comparing the load of the node by a threshold value, if $\text{Load} \geq \text{TT}$ then it is overloaded, otherwise it is LL.
 - ❖ Transfer Interval (TI): it defines the period of collecting information and DM, to determine when the LB mechanism should be operated in each node. The interval should not be too short that will result in increasing the cost of collecting unnecessary information, or too large that leads to lose the efficiency of DM.
 - ❖ Transfer Size (TS): it determines how much of the load should be transferred, when the transfer policy makes a decision of transferring some loads. TS can be too small, which means that the load to be transferred is limited. Or too big,

which means that some of the transferred loads are unnecessary that results on overhead on the communication network.

3) Selection Policy: It determines which load to be transferred, when transfer policy decision is taken. It can be preemptive (it chooses the partially executed tasks to be transferred with its state; it is expensive especially when the state is large), or Non-preemptive (these algorithms choose the tasks that do not begin execution to be transferred, and then there is no state to be transferred with the tasks).

4) Information Policy: It determines when to collect information about the system state, what type, from where, the amount, and how to use the state information to be collected, this policy can be activated in Fixed state (no exchange of state), in Demand driven in process of LB, periodic exchange, or when State changes.

The previous policies must be taken into account altogether, and the suitability of one parameter depends on the other parameters, the goal is to find the appropriate values of them to get an efficient LB mechanism (Kwok, and Cheung, (2004), Chen, et al. (2008), Razzaque, and Hong (2007)).

1.2.4 Load Balancing Parameters

Sharma, et al. (2008), and Gross, (2005), a LB algorithm performance is measured by the following parameters:

- 1) Fault tolerant: it gives the ability of LB algorithms to continue functioning properly, or stop functioning when a failure occurs. If the algorithm stops working, then the degradation of its performance is proportional to the failure seriousness.

- 2) **Forecasting accuracy:** it is the extent of similarity between the actual results that will be created after execution and the predicted values of the calculated results. Static algorithms are better than the dynamic algorithms; regarding this factor.
- 3) **Stability:** this parameter can be described by:
 - ❖ The delay that occurs when transferring information between processing nodes.
 - ❖ The time improvement obtained by the improved LB algorithms, which gives a better performance in a faster amount of time.
- 4) **Centralized or Decentralized.**
- 5) **Nature of LB algorithm:** whether it is static or dynamic.
- 6) **Cooperative:** it determines the degree of independence among the nodes to decide how they will use their resources to allocate processes. In cooperative algorithms, all nodes are responsible for making the decision of scheduling and cooperating to achieve a better performance, each node in the system will use the shared information to make the decision of using its own resources; but in non-cooperative algorithms, each node represents itself independently in DM of how to use its resources without affecting the rest of the system.
- 7) **Process migration:** it determines when to push some load from a busy node to an idle node in order to improve the system performance. These algorithms can make the decision of process execution locally or remotely during the execution.
- 8) **Resource utilization:** it consists of algorithms that are able to use the resources automatically, and hence moving the system from overloaded state to an under loaded state efficiently.

1.3. Thesis Organization

The rest of this thesis is organized as follows: In chapter 2 related works will be presented, our work will be introduced in chapter 3. Simulation environment and the related results taken from the simulation will be presented and evaluated in chapter 4. Chapter 5 gives the conclusions and suggests the future work.

2. Background and Literature Review

2.1 Introduction

The advancement in the communication, and the micro-electronic technologies; causes the availability of efficient, and cost effective computer networks, and the availability of fast, and inexpensive processors respectively. As a result; the price/performance ratio makes it a trend to use a set of loosely coupled interconnected processors to form a distributed environment instead of using single high-speed processor.

In such environments (the DCSs) the probability of having one idle host (has no pending tasks for execution) while other hosts are heavily loaded (has more jobs in its CPU queue are waiting to be executed) is high as proposed by many researches. LB in such environments is considered to be a challenge due to many reasons, such as: a) the autonomy of processors, b) the communication overhead in collecting state information, c) communication delays, and d) overhead in load redistribution.

The goal of LB is to improve the performance and get rid of load imbalance in the system by either transferring the load from the HL nodes to the LL nodes or by initially assigning tasks fairly. Many works has been done for LB in the past years, it has been investigated in different types of computing environments (loosely and tightly coupled systems), using a variety of strategies and at different levels.

For example the algorithms of LB can be Static (the distribution of tasks among nodes is made a priori based on jobs information) or Dynamic (DM of jobs transferring is based on the current load state). The unit of transferring/redistribution can be the whole job or individual processes (part of the job). The transfer policy can be sender initiated (triggered by the HL hosts to search for LL hosts where tasks can be moved to) or receiver initiated (triggered by the LL hosts to search for HL hosts where tasks can be moved from).

The drawback of Static Load Balancing (SLB) that it assumes too much information about the jobs is known a priori (even if they are known, huge computation is needed to form a perfect schedule), make the researchers to prefer the Dynamic Load Balancing (DLB), which make the decision of transferring jobs intelligently during its execution as circumstances changes dynamically. Many papers study this category of LB, in this chapter we will summary most of those closely related to our work.

Brief descriptions of the traditional DLB algorithms are given in (Hac, 1989). Watts, *et al.* (1996), demonstrates that a practical DLB approach is possible by using a framework to implement the algorithm of the LB, called the concurrent graph library.

Chhabra, *et al.* (2006), the authors compare DLB, and SLB algorithms under the qualitative parameters identified by them.

Cao, *et al.* (2000), design a simulator to simulate LB algorithms on a Local Area Network (LAN) of DEC workstations environment, it uses a real workload distribution, and executes the codes of the LB algorithms directly. They use their simulator to compare four simple DLB algorithms.

Game theory is used to analyze and design DSs and SLB methods, by formulating the LB methods in the DS as non-cooperative game between consumers, then transform it to standard convex optimization problem using a function outlined by the authors, (Nouri, and Parsa, 2009).

Broberg, *et al.* (2005), are concerned with another metric of the DS performance, they propose the Task Assignment based on Prioritizing Traffic Flows policy (TAPTF); which focus on reducing the mean slow down metric, in addition to mean waiting time satisfied by the LB policies. This policy differentiates the short traffic flows from the large ones by using dual queues with "cutoffs" (the processing limit associated with

each host), and allow the short tasks to be executed quickly without waiting the large tasks.

All Requests are arrived at the centralized dispatcher queue, the dispatcher will distribute the requests at random in First-In-First-Out (FIFO) manner among the hosts in the DS, Each host except the first one has two queues; the Ordinary queue (OQ), which receives the requests directly from the dispatcher and the Restart queue (RQ), and each of them is assigned a “cutoffs”.

Tasks that exceeds the cutoff of a given host, will be migrated to the next host RQ, and will be executed from scratch, any work is done before migration is lost. Broberg, *et al.* (2006), The same authors of the previous scheme improve it into Task Assignment based on Prioritizing Traffic Flows with Work-Conserving Migration (TAPTF-WC) to decrease the penalty of the non-conserving migration found in other policies, which follow the same method except that task migration is done in Work conserving manner with negligible cost of state information migration.

So when the task exceeds the cutoffs of the host OQ, it will be migrated to with its state information the next host RQ, but in this case any work done before migration will not be lost, the TAPTF-WC will resume execution from the point the task stops before migration, instead of restart executing it from scratch, this process is repeated until the task can be executed to completion.

A new DLB scheme proposed by (Jain and Gupta, 2009), is based on the Centralized approaches and the Interrupt service, they enhance the limitation of having one central node in the centralized approaches by dividing it into smaller nodes. The system in this scheme uses two types of nodes, the primary nodes (the main nodes) called PNs, and the Supporting nodes (used to cope with the overload) obtained by splitting the central node

into smaller nodes called SNs. SNs are assigned some tasks and have a priority queue for their processes.

When a PN is overloaded, then it will search for a lightly loaded PN within its cluster, if it found such node the LB is obtained, otherwise the PN will search for a proper SN, when finding it, the PN will interrupt the SN, then the SN will assign a priority to the coming process, and will call the ISR (Interrupt Service Routine). The ISR will compare the priority of the currently executed process in the SN and the coming process, and will take the decision of process execution.

A hierarchical approach that is based on two phases to collect information and make the decision of LB is adopted. 2D load index (job attribute) is incorporated in DM, which is called the job size, (Vaughan, 1995).

Two hierarchical DLB algorithms are presented by (Barazandeh, and Mortazavi, 2009), the first algorithm called dynamic biasing; depend in distributing the load among the nodes according to their weights. Weights allocated to nodes and groups based on their current load state. The second algorithm is an enhancement to the RR algorithm; it assigns tasks to the node with minimum load state in a certain time slice. This algorithm is called Minimum Load State Round Robin (MLSRR).

Simulations that perform routines and repetitive computations on a large data sets are considered, especially when these iterations are irregular in their nature. Three hierarchical Dynamic Loop Scheduling (DLS) approaches are proposed to maximize the performance of such applications (Banicescu, *et al.* 2009).

A traditional Decentralized D LB scheme for a homogeneous DS called state collection algorithm, was addressed by (H.Ammar and, Su Deng, 1988). It assumes that each node has a state record of all other nodes, initially; the values of the length of CPUs' jobs queues of all other nodes in the state record are zeros, each node sends its own state

when it sends jobs to the other nodes, the decision of Sending jobs depend on comparing the job's queue length of the sending node and the other nodes, if the number of local jobs exceeds the number of remote jobs, then a sending decision must be taken to a randomly destination node if the state record is empty.

After further sending operations, the state record will be built by updating it each time a new job is received with the sending node state, in this situation the decision is taken according to the state record, by choosing a destination node with the minimum number of jobs in its queue.

Periodic Symmetrically-Initiated (PSI) LB algorithm is presented, which is a hybrid of four LB mechanisms, namely, Random, Sender, Receiver, and Symmetric Algorithms. (Benmohammed-Mahieddine, and Dew, 1994)

The behavior of the workload, including the process life time distribution, effects the effectiveness of LB as proposed by the authors, which is used to derive a preemptive migration policy. They reported that while Preemptive policies have a higher migration cost, it performs better than non-preemptive policies which are less efficient. They use the distribution of process life time to calculate the minimum age of the process for migration, which improves the expected slowdown of the process, (Harchol-Balter, and Downey, 1997).

Keqin Li, (1998), outlines a new scheme in which there are many dispatchers, one on each computer in the system, and each of them receives a separate task generation streams. The dispatchers assigns tasks to computers without any knowledge of the load status information and without any coordination between them to balance the load among all the machines, in such a way that whenever a task is assigned to a certain

computer, it will never be migrated to another computer, so no need to operate LB, because it is achieved together with task assignment.

Heuristic methods for process migration and threshold update are used by (Xu, and Hwang, 1990) in message-passing multicomputer. Heuristic methods are also used to balance the load in a Heterogeneous DS, the proposed algorithm works in two phases; the Recursive Neighbor Search Algorithm (RNS) in the first phase finds the most suitable node that minimizes the task response time in its neighborhood, based on the local information of the neighborhood.

The second phase, which is The Augmented Tabu-Search Algorithm (ATS), is triggered only when the system is out of an efficient Load Balance threshold because, operating it in all cases results in additional overhead to the system. ATS is triggered to more balance the system load, and to overcome the possible weaknesses and bottlenecks of the RNS (Savvas, and Kechadi, 2004/b).

Irregular topologies are considered in many schemes to balance the load on them, these schemes based on mapping them to a regular topology, such as, hyper-grid, tree, grid, etc. for example a new Dynamic task scheduling mechanism for computing clusters is proposed, called the Positional Scan Task Scheduling (PSTS) that is based on the Positional Scan Load Balancing algorithm (PSLB). The PSTS algorithm is operated in two phases based on "divide and conquer" principle.

In the first phase the network (possibly irregular) is mapped into a hyper-grid topology, and then in the second phase the load will be re-distributed among the nodes by dividing the hyper-grid (of dimension K for example) into hyper-grids of smaller dimension ($K-1$). then the load balanced hyper-grids of dimension $K-1$ will be divided again, this

process of dividing the hyper-grids will be recursively continued until their dimension is equal to 1, (Savvas, and Kechadi, 2004/a).

In Akhtar, and Kechadi, (2006), the P2P computing system is mapped to a TreeP topology², and then the workload will be redistributed among the peers by using the PSLP algorithm based on their PS and their current loads.

Two SLB approaches are reviewed to propose two DLB schemes; The Dynamic Global Optimal Scheme (DGOS), and Dynamic Non-CoOPERative scheme with Communication (NCOOPC) for multi-class jobs in heterogeneous DSs. DGOS tries to minimize the expected response time of the whole system while the other scheme tries to provide a user optimal solution by minimizing the expected response time of individual users, (Penmatsa, and Chronopoulos, 2007).

The heterogeneous DS is embedded onto a structure similar to B+ tree first, and then the Enhanced PSTS (EPSTS) is applied to balance the load in the virtual structure, (Savvas, and Kechadi, 2007).

Barbosa, and Moreira, (2009), improve the performance of a distributed memory computer; in their proposed scheme, each job in their algorithm is described by a Direct Acyclic Graph (DAG), and composed by a set of dependent working units. They enhance the common approaches which assign the entire job to a single processing node, by scheduling the working units³ of tasks from all tasks dynamically.

They use the batch strategy in their scheduling, which consider the new tasks and the previously scheduled tasks, but waiting for service, Two algorithms are considered for

² TreeP: it is a hierarchal topology based on 1-D space tessellations (tree based P2P architecture).

³ Working units are considered the basic processing unit, instead of considering the task as basic element.

scheduling DAGs in their approach, namely the Heterogeneous Earliest-Finish Time algorithm (HEFT) and the Heterogeneous Parallel Task Scheduler (HPTS).

Wang, et al. (2009), proposed an adaptive LB mechanism within the framework of middleware. This mechanism is based on machine learning to predict the load fluctuations, and react to them gradually, and uses the fuzzy logic to manage the replica.

Razzaque and Seon Hong (2006) proposed a D LB scheme applied on a homogeneous DCS that is between the centralized and the distributed approaches, the performance metric that is considered here is minimizing the communication cost evaluated by the total number of messages exchanged in the communication network, and reducing the turnaround time of load execution compared with the common-used schemes.

The system of N nodes is divided into N different mutual overlapped subsets of size K , (where $K=\sqrt{N}$ approximately), by using Maekawa's algorithm, such that each subset overlaps every other subset. Each node is assigned to a specific subset (called the request set), and can request the state information from only the members of its request set and exchange information with them, to update the system state table and make the decision of scheduling to decide whether to execute the load locally or remotely, and how much load can it send (to) or receive (from) the other nodes.

The solution of Incremental Weight migration (IWM) problem on arbitrary graphs is used to derive a DLB algorithm. IWM uses the matching of a random set of edges to balance the load of the system. (Ghosh, and Muthukrishnan, 1994)

Perez, (1997), present a method that uses the definition of High Performance Fortran (HPF) to constitute the LB method into the run time, which uses the virtual processors as basic unit of migration in the LB mechanism. He controls the distribution of them onto physical processors to balance the load of the system.

Another way to cope the problem of LB in a DCS is proposed here, instead of using a D LB algorithms, the author tried to minimize the probability of LIB (Load ImBalance), to reduce the overhead needed to call a specific DLB algorithm, which forces the system to process migration, so he try to find an optimal load distribution to avoid using LB mechanisms. The load are distributed in such a way, that the nodes with a higher PSs takes more load than the nodes of the lower PSs.

He assumes that a DCS is composed of two levels, a probabilistic job dispatcher on the first level, and a DLB algorithm on the second level, the probabilistic job dispatcher adjusts the job arrival rate into a specific load distribution. The state of a node is described by the current number of jobs in its queue (either they are waiting in the or being executed).

Two thresholds are used U , and L , to classify nodes' states, whether it is HL, LL, or Moderately Loaded (ML). If load imbalance occur (if having at least one under-loaded node and one overloaded node in the DCS), a D LB algorithm in the second level will take place to migrate some loads from the busy node to the idle node, (KEQIN LI, 2002).

Fedorov, and Chrisochoides, (2004), design and implement a run time system called CLAM. It provides communication support for DLB of irregular adaptive applications. Chen, *et al.* (2008) investigate the contribution of the evolutionary learning algorithms to solve the problem of LB, they consider the evolution as one of the basics methods to solve problems by nature.

They study the effect of the following parameters in controlling the LB in a fully connected DS; a) Increasing the network heterogeneity (including the PS of nodes, and Transmission speed), b) Network topology (star or ring), and c) {TI, TS, and TT}

parameters, the authors pointed out finding the appropriate values of all parameters is needed in order to get an effective LB mechanism by using the evolutionary learning.

The impact of statistical properties of the network on the performance of LB is studied by (Fukuda. *et al.* 2007); they discuss the effectiveness of using the topological information on the performance of the Multi-Agent System (MAS), in term of coordination, because it provides the environmental information about the world for the agents.

They investigate a deployment algorithm, that put each agent in the proper position in the internet at design step, then they propose selection algorithm (to select the appropriate agent to collaborate with) that needs only the degree and the scope parameters, without global information about the network structure to balance the load (i.e., achieve fairness) in the MAS.

A fair and DLB mechanism that considers the needs of both the user and the operator (who is interested in the communication between the of the Origin-Destination (OD) pairs) is presented, they define a utility function of the OD pairs; whose arguments is the average available bandwidth seen by each OD pair path, and maximize the sum of all OD pairs, but the formulated optimization is not convex, they solve this problem by using a Distributed algorithm, (Larroca, and Rougier, 2009).

2.2 Related Work

Barazandeh, and Mortazavi, (2009), offer a dynamic algorithm for LB in DSs called Dynamic Biasing Algorithm (DBA) that is based on Hierarchical structure. In this algorithm, the workload is distributed among the nodes of the DS according to some values called biases, which are determined based on the current load of nodes and groups. The underlying DS has hierarchical structure as shown in Figure 2.1.

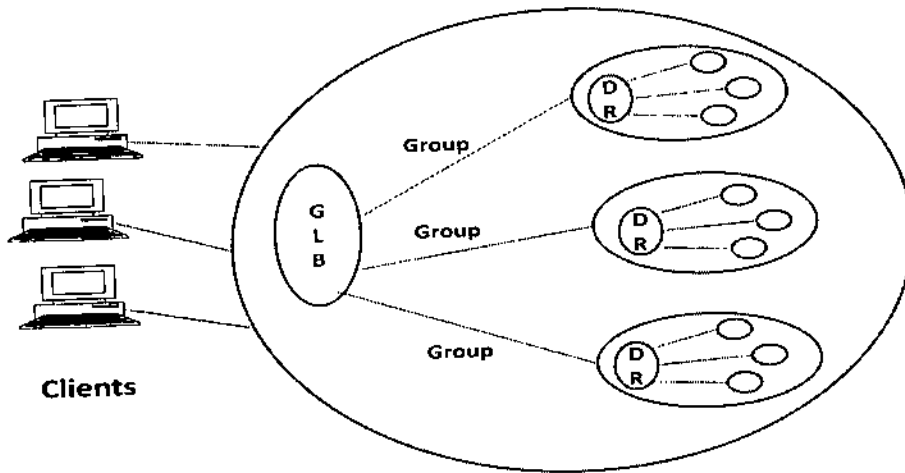


Figure 2.1: Underlying Distributed System

In this scheme the LB in the DS is executed in two phases; in the first phase a node called Designated Representative (DR) or the Local Load Balancer (LLB) is responsible for the local load balancing among nodes within the group. In the second phase the Global Load Balancer (GLB) balances the load among the groups. The LB operation is centralized in both phases.

As the underlying structure shows the DS consist of a number of groups; each group has its own DR that is connected to both, GLB and to its group nodes. The groups are not connected to each other, means that when a task is allocated to a specific node in one of the groups it must be processed in that group, and no replacement is permitted.

Each node sends its current load state (CPU Queue length) at a specific time ,called state checking times, to its group's DR, then the DR based on the information received from nodes will do biasing, which is the process of giving weight or bias for each node, and do local load balancing among nodes. The same process is repeated between the LLBs which calculate their group load state and send it to the GLB, which in turn will do biasing and load balancing among groups.

Bias is calculated as follows: the nodes of one group are sorted according to their current load state, and then based on the group load state and the number of tasks in the LLB; the bias of each node will be some percent of these tasks. The LLB will distribute tasks among nodes according to their biases. The GLB will calculate biases for each group according to group load state, and the number of tasks that exist in GLB memory.

Nodes and DRs send their current load state and group load state to their DR and GLB, respectively. This process is done in each checking state time without any request from the DR and the GLB, because the pushing policy is used for exchanging and updating information in the system. The intervals of doing biasing and state checking times are chosen carefully.

3. Enhanced Dynamic Biasing Algorithm for Load Balancing in Hierarchical Distributed Systems

3.1. Overview

In this chapter, we will explore in details the proposed scheme which hereinafter called Enhanced Dynamic Biasing Algorithm (EDBA), the enhanced scheme aims to increase the performance of the DS by increasing the throughput of the processed tasks, also in decreasing the response time. Generally, it's known that, the problem of finding optimal solution of the Load Balancing is NP-complete.

The DS with hierarchical structure is introduced because of the following advantages:

- Provides a better management of the workload because the load balancing among groups and nodes is done in a number of phases.
- Decreases the communication overhead among nodes because of using the centralized policies of LB.
- Provides a better performance for a large DS if it is compared with distributed LB policies which incur a huge number of messages to exchange the load state among nodes.
- Also in a large DS, messages from remote nodes will have significant propagation delay which makes the arrived load state out of date.
- Ability to scale large DSs.

In the other hand, the dynamic LB algorithm is introduced, which made decisions of distributing load among nodes intelligently based on the current load state of nodes that changes dynamically during time. Although, it has complication in implementation and higher communication overheads, it outperforms the static algorithms that are simple to

implement and have low communication overhead. Static algorithms distribute load among nodes randomly and doesn't use the current load states of nodes. Exchanging load state messages among nodes in static algorithms is no longer needed.

Two improvements are made to the original algorithm (DBA), the first improvement is the EDBA with CPU Utilization (EDBA-CPU-U), which increases the system performance. The other improvement is EDBA with process migration (EDBA-PM), which provides a better performance compared with DBA.

In EDBA-CPU-U a new parameter is used to calculate the biases of nodes and groups, this parameter is the CPU Utilization (CPU-U) rate of each node. It's known that low CPU-U means the node is idle in most of the time, while high CPU-U means the node is working properly and it is busy in processing tasks most of the time. our aim is to minimize nodes idling time and make them busy most of the time in order to increase the performance of the system, in terms of throughput and accordingly response time.

In EDBA-PM, the process migration policy is used in the DS when a node load is higher than a specific threshold, then some of its load will be transferred to another lightly loaded node. This enhancement increases the system performance very good when task arrival times are variable, and unpredictable compared with the original algorithm.

3.2. The System Model

The underlying structure of the DS is hierarchical, and the DS could be homogeneous or heterogeneous. In homogeneous DS all nodes have the same service rate and memory; while in the heterogeneous DS, the nodes have completely different memory and service rate.

The DS has a tree structure where the system consists of two groups, each group has eight nodes, and the system has three clients that generates tasks and send them to the system. The architecture of the system is as follows: the LLBs are connected to the GLB, and the nodes are connected only to their LLBs. Nodes are not connected to each other; they can communicate with each other only by their LLB (this communication among nodes is needed only in the EDBA-PM). All groups are connected to the GLB by their LLB, and there is no connection between groups.

The groups that compose the desired DS have close communication property, where the nodes of each group communicate with the nodes of the same group and can't communicate with the nodes of the other group. As shown in figure 3.1 there are no communication links among the nodes of different groups. They can communicate with their LLB only.

The Load Balancing is operated in two levels as in the original algorithm, the LLB balances the load among the nodes of its groups, then the GLB balances the load among the groups, but here in the presented scheme the LLBs and GLB do load balancing in a different way from the DBA, in such a way, that it provides a better system performance. The figure 3.1 shows the DS considered in the proposed scheme.

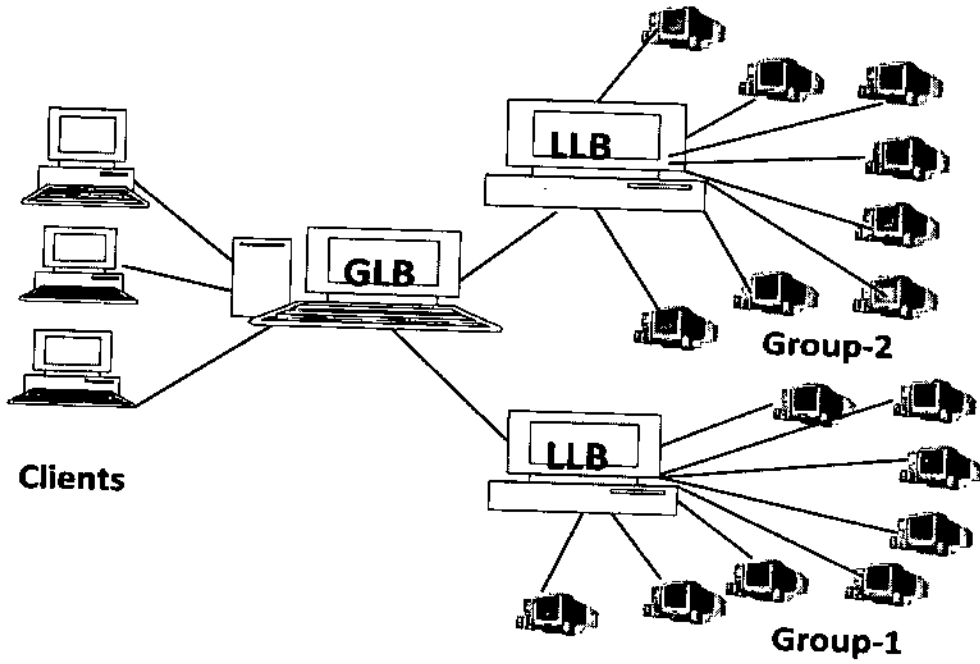


Figure 3.1: Underlying Distributed System Structure

3.3. The Proposed scheme Assumptions

This study concerns about homogeneous and heterogeneous DSs of Hierarchical structure with these assumptions:

- Groups are already formed.
- This study doesn't consider how nodes join or leave groups. ⁴
- All nodes have the same memory and the same service rate in homogeneous system.
- All nodes have completely different memory and different service rate in heterogeneous system.
- The communication links have some delay.

⁴ For interested readers, groups forming can be done in many ways, (Ahn, *et al.* 2007), (Barazandeh, *et al.* 2009), and (Huang, *et al.* 2003).

3.4. Determining Suitable time intervals

The process of calculating biases is called biasing; this process is done periodically in time interval called biasing interval. Biasing interval must be chosen efficiently in such a way, it must not be too high which increases the response time and makes the algorithm not efficient. In the other hand, low interval will increase the response time because several zero biases will be made and the tasks will remain in the LLBs and in the GLB, so this interval must be chosen very carefully.

In this scheme there are several choices for this interval, we evaluate the performance of the system with each one of the previous choices; then we select the time interval that gives the best performing system.

As in the original algorithm pushing policy is used to exchange information among nodes and LLBs, and between the LLBs and the GLB; meaning that the nodes send their current load state information in each checking state time without any useless request from the LLBs. The LLB will send the group current load state to the GLB according to the received information without any request from the GLB.

Pushing policy increases the system performance and decreases the communication overhead, because it omits the useless request messages of the current load state.

The state checking time must be chosen very carefully, because high interval makes the incoming information about the load state out-of-date, as a result; the decision of the load balancing will be invalid. Low interval will increase the number of exchanged messages which increases the communication overhead and degrade the system performance.

In the proposed scheme many values are put as state checking time, we evaluate the system performance with all of them, and then we choose the time interval that gives the best system performance.

3.5. Enhanced Dynamic Biasing Algorithm with CPU Utilization (EDBA-CPU-U)

In this scheme a new load index is used to calculate the nodes biases, in order to enhance the system performance; this index is the CPU-U rate of each node. So, instead of sending the CPU queue length periodically to the LLB as in the original algorithm, the node will also calculate its CPU-U rate and send it accompanied by its queue length in each state checking time.

The aim of this enhancement is to keep all nodes having high CPU-U at all times, and attempt to balance the load in the DS by distributing tasks evenly among the nodes; each one will have some percent of all tasks based on its current load state and CPU-U rate. Maintaining high processor utilization in nodes can be achieved by keeping the nodes busy at all times and minimizing their idling times.

According to this scheme; the node that has lower CPU-U rate will receive a larger share of tasks than the other nodes; making it busy for longer time in processing the incoming tasks, and increasing the system performance accordingly.

3.5.1. Methodology

The EDBA-CPU-U algorithm involves the following phases:

- 1) Load Evaluation: each node computes its current load state in percentage unit, which includes its CPU queue length, and its CPU-U rate as in Equ(1):

$$\text{Node-load-state} = (\text{CPU-Q-length} + \text{CPU-U}) / 2$$

Equation (1): Load Evaluation

Where CPU queue length and the CPU-U rate is calculated as follows:

- CPU queue length: refers to the percentage of the number of tasks that are in the node's processor queue, waiting to be executed by the CPU to the whole queue size including the free size and the occupied size (where queue size is measured by the number of tasks that it can save).
- CPU-U rate: the percentage of the busy time of the processor to the whole processor time, for example if the processor starts working at time t1, then it becomes busy at time t2, and finishes working or becomes idle at time t3; then the CPU-U rate is calculated as follows:

$$\text{CPU-U} = [(t3-t2) / (t3-t1)] * 100\%$$

- 2) State Information exchange policy: each node periodically sends its current load state to its LLB, and each LLB sends its current group state to the GLB. This interval of sending load state called state checking time, the group state is calculated as in Equ(2):

$$\text{Group-load-state} = \sum \text{nodes-load-state} / \text{num-nodes.}$$

Equation (2): State Information exchange policy

- 3) Biases Calculation: each LLB calculate its group nodes' biases based on the received information about the current load state from nodes, and each GLB calculates groups' biases based on the received information of the current group

state from the LLBs, the biases of nodes calculation is done as in equ(3); and the biases of the groups are calculated in the same way.

$$\begin{aligned} \text{Bias [node-i]} &= 100 - \text{node-load-state [node-i]} \\ \text{Sum} &= \sum \text{Bias-all-nodes} \\ \text{Bias [node-i]} &= (\text{Bias [node-i]} / \text{sum}) * 100 \\ \text{Bias [node-i]} &= (\text{Bias [node-i]} / 100) * \text{LLB-Q-Length} \end{aligned}$$

Equation (3): Biases Calculation

- 4) Task Distribution: tasks are distributed among groups and nodes according to their biases, based on the number of tasks in the GLB and groups biases, each group will receive a specific percent of the tasks, then according to the number of tasks received from the GLB and nodes biases; each LLB will distribute these tasks among nodes, each with some share of the tasks.

3.6. Enhanced Dynamic Biasing Algorithm with Process Migration (EDBA-PM)

In this scheme the process migration policy is used to enhance the system performance by transferring some load from the heavily loaded nodes to the lightly loaded nodes, so all nodes will have at least one task to process at all times. This enhancement increases the system performance with respect to throughput, and response time parameters; especially when the task arrival rate is variable and unpredictable.

Task migration is typically justified by the assumption that the migrated task will be processed faster on the resources on the other node, this assumption based on the load measure comparison between the source and the destination node.

3.6.1. Methodology

The EDDBA-PM algorithm involves the following phases:

- 1) Load Evaluation: each node computes its current load state, which includes its CPU queue length as shown in Equ(4):

$$\text{Node-load-state} = \text{CPU-Q-length}$$

Equation (4): Load Evaluation

- 2) Each node will check if its current load is greater than some threshold T, and take the decision of task migration as Equ(5):

```

If (Node-load-state >= T)
    Then: Migrate Task (J) to LLB;
Else
    Don't migrate any task;
  
```

Equation (5): Decision of process migration

- 3) State Information exchange policy: each node periodically sends its current load state to its LLB, and each LLB sends its current group state to the GLB. This interval of sending load state called state checking time, the group state is calculated as Equ(6):

$$\text{Group-load-state} = \sum \text{nodes-load-state} / \text{num-nodes.}$$

Equation (6): State Information exchange policy

- 4) Biases Calculation: each LLB calculate its group nodes' biases based on the received information about the current load state from nodes, and each GLB calculates groups' biases based on the received information of the current group state from the LLBs, the biases of nodes calculation is done as follows; and the biases of the groups are calculated in the same way.

$$\begin{aligned} \text{Bias [node-i]} &= 100 - \text{node-load-state [node-i]} \\ \text{Sum} &= \sum \text{Bias-all-nodes} \\ \text{Bias [node-i]} &= (\text{Bias [node-i]} / \text{sum}) * 100 \\ \text{Bias [node-i]} &= (\text{Bias [node-i]} / 100) * \text{LLB-Q-Length} \end{aligned}$$

Equation (7): Biases Calculation

3.6.2 Process migration policy used in the Enhanced scheme

The process migration policy used in the enhanced algorithm composed of three phases to make the decision of task migration, these phases are described below:

- 1) When to initiate task migration: this phase determine whether the task should be executed locally or remotely. in this phase at each task arrival the node checks if its current load is greater than some threshold T , if the condition is satisfied then this node is heavily loaded, and won't insert this task to its CPU queue and will migrate it, otherwise the task will be inserted to its queue and will wait to be serviced.
- 2) The where phase: it is concerned with which node will be selected as a destination node to execute the transferred task. This phase is done totally in the

LLBs. When the node is heavily loaded; it will migrate some tasks to its LLB, and then the LLB will choose the destination node that will execute this task according to nodes biases.

- 3) Which phase: it determines which task will be selected to be migrated to the LLB when the decision of task transferring is made. We have decided to migrate the older tasks, because these tasks have the probability to live for longer time enough for amortizing the cost of process migration, (Campos, and Scherson, 2000)

Two observations must be made to the first phase. First, each node will check the previous condition at different time interval, it is not necessary that all nodes will receive tasks at the same time. Second, the threshold is not constant and changes adaptively according to the current load state of the node and may change from one node to another.

The transfer threshold which determine when to consider the node as heavily loaded or lightly loaded must be chosen efficiently; a node with high T value indicates that it will keep most of the load assigned to it and try to execute it locally instead of push it to other nodes, while low T value indicate that the node will push most of its load to the other nodes; in both cases the performance of the system will degrade.

In scheme we choose a suitable T value that makes the system performs efficiently, this T is adaptive and differ from one node to another, and it is different from one simulation case to another.

The process migration policy used here is dynamic, distributed, and non-preemptive. Dynamic because it satisfies the changing requirement and make the decision of task migration depending on the current load state of nodes, Distributed, because the

decision of PM is made locally by each node regardless of the other nodes, and non-preemptive, because tasks are migrated before executing them.

3.7. Simulation Cases

The presented scheme studies Dynamic Load Balancing in Distributed System with twelve different cases, The DLB are studied under homogeneous and heterogeneous DS; with homogeneous DS all nodes have the same memory and service rate, while in the heterogeneous DS all nodes have completely different memory and different service rate.

At both DSs, three algorithms are evaluated under constant arrival rate of tasks in which the time between the first arrived task and the next one is fixed and known, and under variable arrival rate of tasks, in which the tasks inter-arrival time is variable and unpredictable. The exponential distribution with mean equal (0.5) is used to represent variable inter-arrival time.

The three algorithms are the original algorithm, Dynamic Biasing Algorithm, the Enhanced Dynamic Biasing Algorithm with CPU-U, and Enhanced Dynamic Biasing Algorithm with Process Migration. These cases are summarized in Figure 3.2.

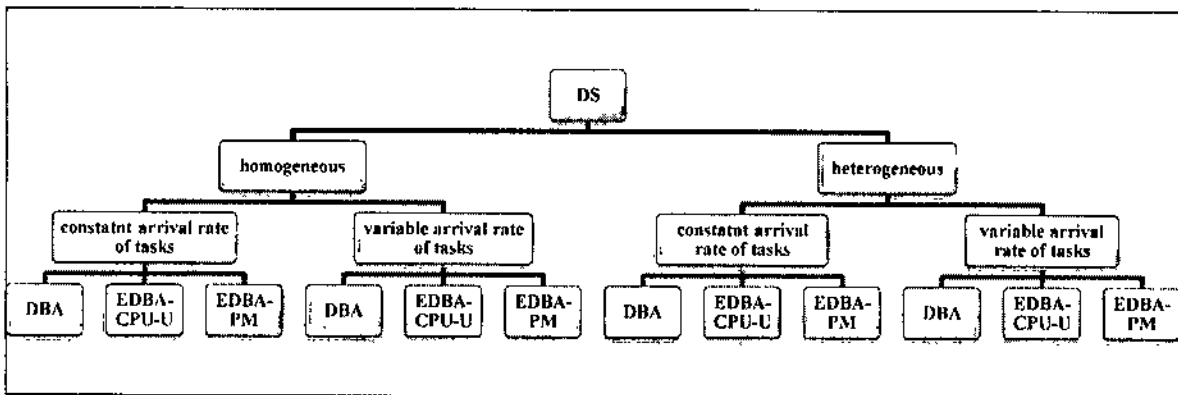


Figure 3.2. Simulation cases

4. Results and Evaluation

4.1. Introduction

In this chapter, we will present the simulation process we followed in details. To start with, an overview of the simulator used in our experiments will be given. Next, different scenarios will be discussed along with their simulation setup parameters. To evaluate the performance of our methods, certain metrics should be chosen to give us the right indication about our goals, these metrics will be explored. Finally, simulation results along with their analysis will be given.

4.2. The Simulator

Our schemes EDBA-CPU-U and EDBA-PM have been simulated using network simulator called OPNET.Modeler.14.0.A.PL3- Education Version.

OPNET Modeler accelerates the Research &Development of process for analyzing and designing communication networks, devices, protocols, and applications. Users can analyze simulated networks to compare the impact of different technology designs on end-to-end behavior. Modeler incorporates a broad suite of protocols, and technologies, and includes a development environment to enable modeling of all network types and technologies including: VoIP, TCP, OSPFv3, MPLS, IPv6, and Others. [Source: http://www.opnet.com/solutions/network_rd/modeler.html].

We ran the simulator on a computer that has the following capabilities: Intel(R) Core(TM) 2 Duo CPU, T7500 @ 2.20GHz (2 CPUs) Processor, 3GB of RAM Memory, and 160 GB Hard Disk.

The enhanced algorithm is programmed in the node level of the simulator, where each node has a process model that is composed of Finite State Machine (FSM). The FSM consists of many states and transitions that perform the conditions among states, and the

code inside each state represents the serial events of the algorithm. Figure 4.1, 4.2, and 4.3 represents the node level of EDDBA-CPU-U, the process model of the LLB, and the code of one state of the FSM, respectively.

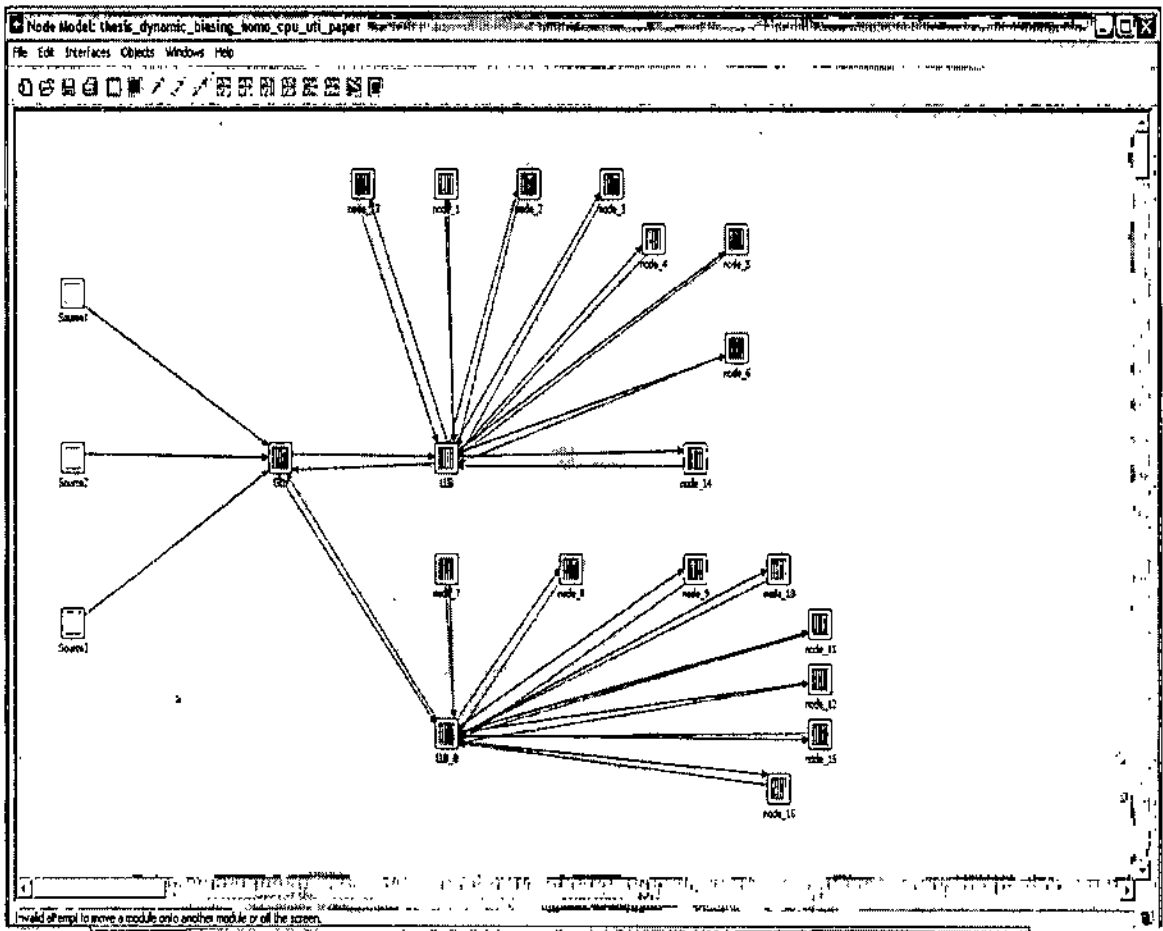


Figure 4.1 The Node level of EDDBA-CPU-U Algorithm.

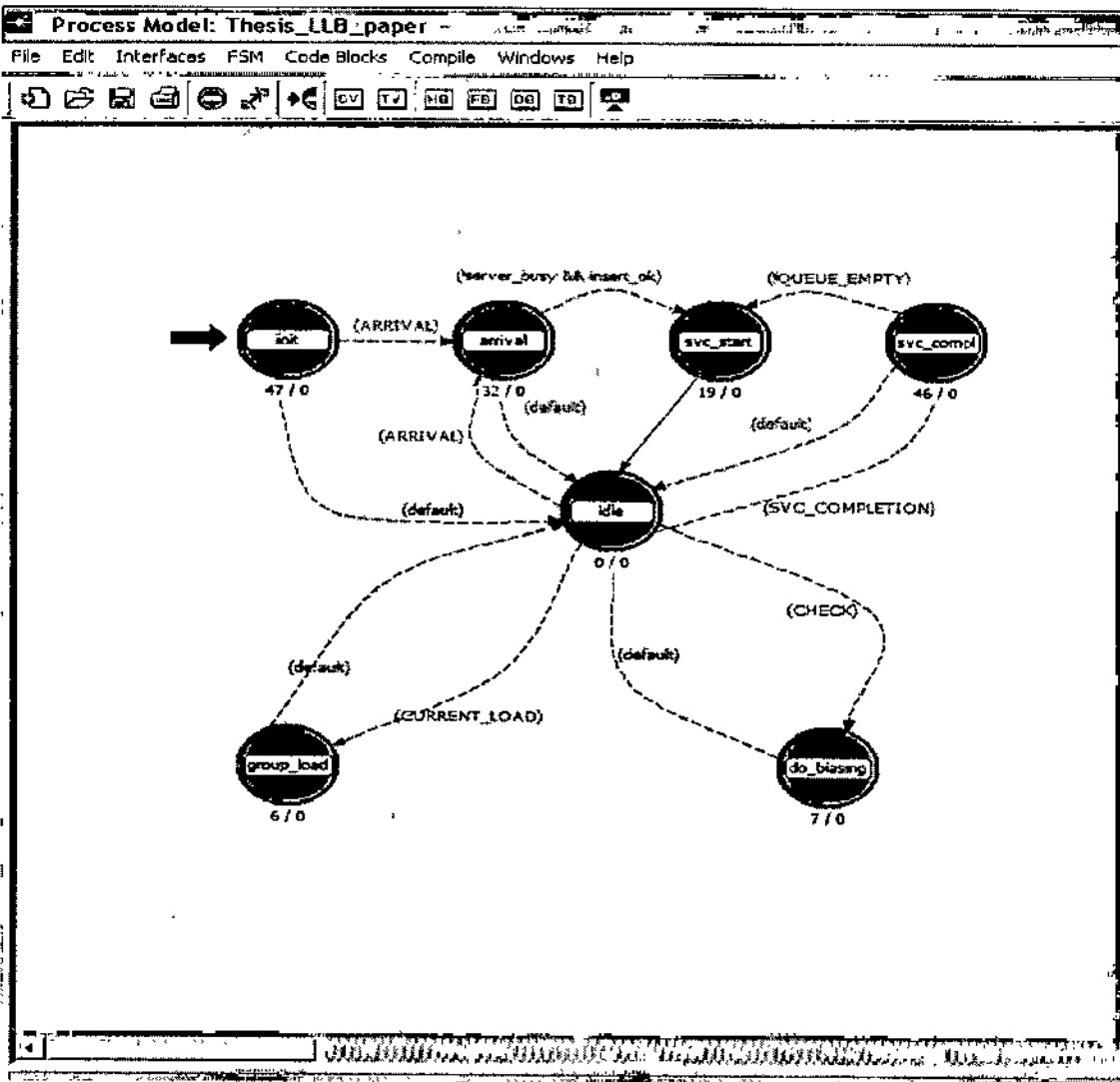


Figure 4.2 The Process Model of LLB.

```

Thesis_ILB_paper : arrival : Enter Exec
File Edit Options
1 /* acquire the arriving packet */
2 /* multiple arriving streams are supported. */
3
4 pkptr = op_pk_get (op_intrpt_strm ());
5
6 /* attempt to enqueue the packet at tail of subqueue 0 */
7
8 if (op_subq_pk_insert (0, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK)
9 {
10 /* the inserton failed (due to to a full queue) deallocate the packet. */
11
12 op_pk_destroy (pkptr);
13
14 /* set flag indicating insertion fail */
15 /* this flag is used to determine */
16 /* transition out of this state */
17 insert_ok = 0;
18 }
19 else
20 {
21 /* insertion was successful */
22 insert_ok = 1;
23 all_pk++;
24 }
25

```

Figure 4.3 the code of the arrival state of the FSM.

4.3. Scenarios of the Simulation

We divided the simulation study samples into two groups; this helped us to trace different scenarios and conditions. The first group of simulations done for homogeneous DSs, the other group tested heterogeneous DSs. Inside each group we divide the simulations depending on the arrival rate of tasks whether it's constant or variable. These simulation cases are summarized in Figure 3.1 in the previous chapter.

4.4. Performance Metrics

As explained in the previous chapter the value of the state checking time, in which the node calculate its current load state, and the biasing interval in which the LLBs and the GLB calculates the biases of the nodes and groups, respectively; is very important, because it affects the system performance very intensively. As said before, these values are chosen based on the simulator results; we try a number of values and then we choose the value that gives the best performance of the system. The chosen values for these intervals are shown in Table 4.1:

Table 4.1: Determining Simulation Intervals

Type of Element	State Checking Time	Biasing Interval
<i>Node</i>	20 seconds	No-Biasing
<i>LLB</i>	22 seconds	25 seconds
<i>GLB</i>	No-State-Check	15 seconds

4.5. Results and Discussions

To compare the proposed schemes with the original one, we develop several simulation cases for different situations. To compare between two algorithms, we calculated the percentage difference as in Equ (9):

$$\%Difference = \frac{|Average (Algorithm2) - Average (Algorithm1)|}{Average Algorithm1} * 100\%$$

Equation (9): Percentage Difference

Where: algorithm1 is the original algorithm, and algorithm2 is the proposed algorithm.

The DBA algorithm and the proposed algorithms are compared in terms of throughput,

and response time. Throughput metric is the number of completed tasks per unit time, and the response time is the time between submission of a specific task in the queue and the first response to that task.

4.5.1. Homogeneous Distributed Systems

We start by simulating the homogeneous DSs, in which all nodes in all groups have the same service rate and same memory capacity. The service rate of the nodes is set to 70 bit/second, and 50 bit/second to the LLBs and the GLB. The communications links that communicate nodes and groups have some delay. The simulation time is set to one hour.

4.5.1.1 Constant Arrival Rate of Tasks

In this simulation case, the task arrival rate is constant, means that the number of the arrived task is know, and the time between two consecutive tasks is constant. As shown in Figure 3.1, which shows the underlying Distributed System structure, there are three task generators, we choose a constant generating rate for all of them, each one can generate from zero to ten tasks a second. The drop rate is not considered when the arrival rate is constant, because it is almost zero.

4.5.1.1.1 EDDBA-CPU-U

The CPU-U rate of the nodes is calculated each time the node checks its current load state; it adds it to the CPU queue length then sends it to its LLB. Figure 4.4 shows the comparison between the DBA and the EDDBA-CPU-U in terms of response time.

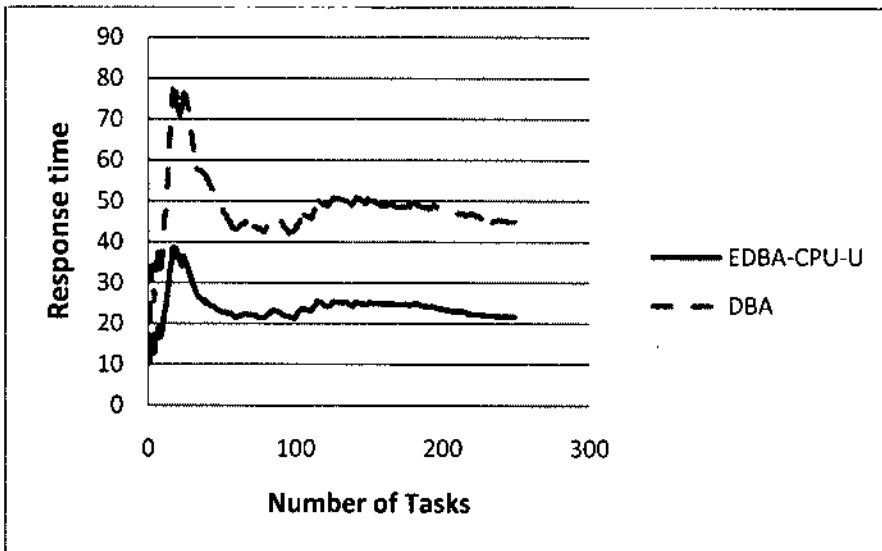


Figure.4.4 Response time with varying number of tasks for EDDBA-CPU-U with constant arrival rate of tasks in homogeneous DSs.

As it can be seen in figure 4.4 the EDDBA-CPU-U outperforms the DBA in terms of response time by 45% approximately. The DBA and the EDDBA-CPU-U are compared with varying number of tasks. The enhanced algorithm decreases the response time very much because, when the CPU utilization rate is considered to distribute tasks among nodes; the idling time of nodes will decrease and the response time will be improved as a result.

In the previous case the number of tasks that are received by each node equal 250 tasks in average, and the total number of tasks that are received by all nodes equal 18,000 tasks approximately. The number of processors that are used to execute the tasks equal sixteen processors. Figure 4.5 shows the comparison between the two algorithms but with varying number of processors.

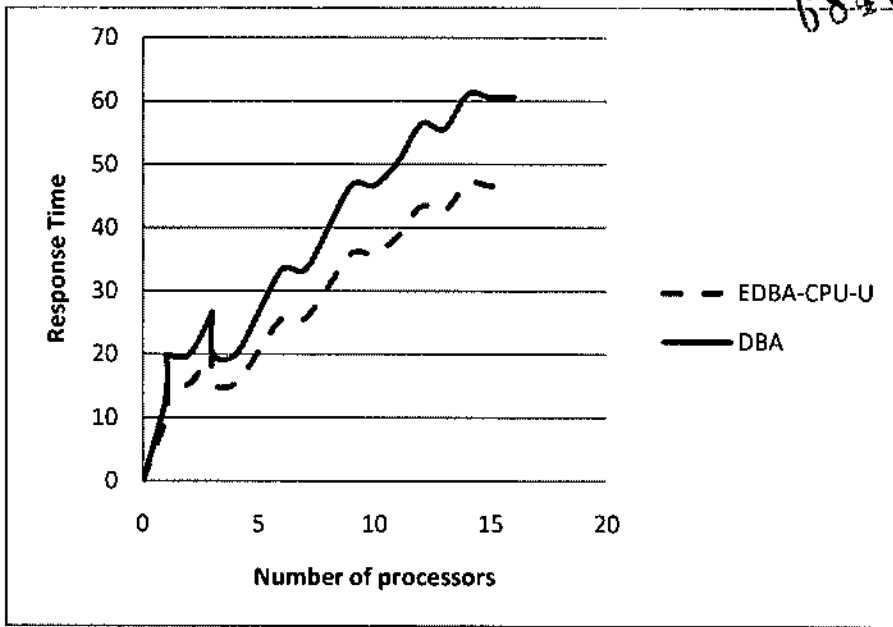


Figure 4.5 Response time with varying number of processors for EDDBA-CPU-U with constant arrival rate of tasks in homogeneous DSs.

The previous figure compares between the DBA and the EDDBA-CPU-U in terms of response time, where the arrival rate of tasks to the nodes are constant and with varying number of processors. The numbers of processors used are sixteen processors.

We can note that the proposed algorithm increases the performance of the system in terms of response time by 31.07%. This result is expected because of using the CPU-utilization as load index in addition to the queue length to determine the node load state will increase the system performance and will gives better results.

4.5.1.1.2 EDDBA-PM

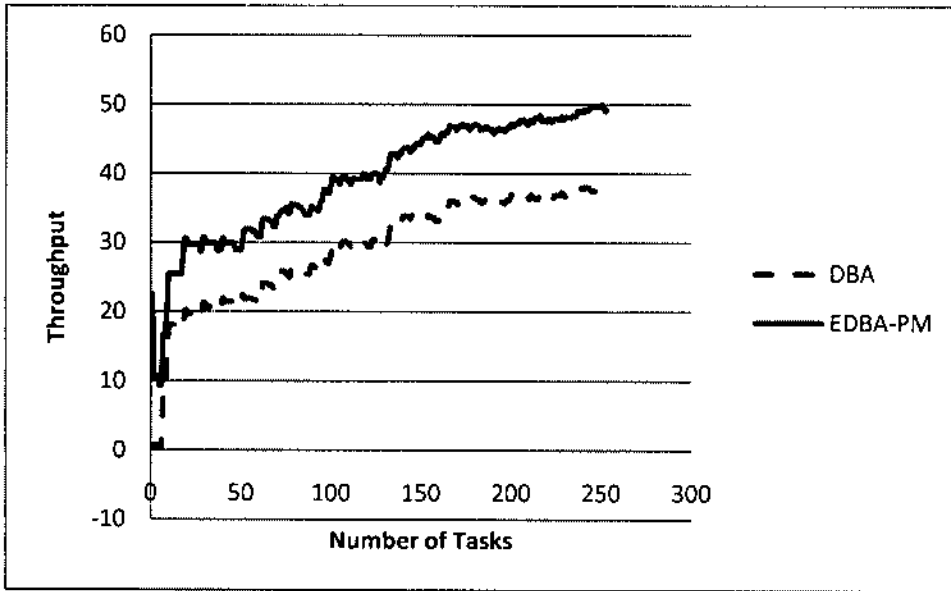


Figure 4.6 Throughput with varying number of tasks for EDDBA-PM with constant arrival rate of tasks in homogeneous DSs.

The previous figure shows the difference of the two algorithms performance with respect to throughput factor. It is clear that the proposed algorithm has a better performance than the DBA algorithm by 15.5% approximately. This percentage is not large because, as we said that the EDDBA-PM enhances the system performance by a larger percentage when having variable arrival rate of tasks.

But in general the EDDBA-PM increases the number of processed tasks because it tries to balance the load among the nodes in the DS, and make them have the same number of tasks approximately by using the process migration policy. This policy enhances the system performance by transferring some load from the heavily loaded nodes to the lightly loaded nodes.

In the previous case the number of tasks that are received by each node equal 250 tasks in average, and the total number of tasks that are received by all nodes equal 18,000

tasks approximately. The number of processors that are used to execute the tasks equal sixteen processors.

Figure 4.7 describes the difference in the two algorithms performance with varying number of processors, as the graph depicts the EDBA-PM outperforms the DBA in terms of throughput factor by 24.85%.

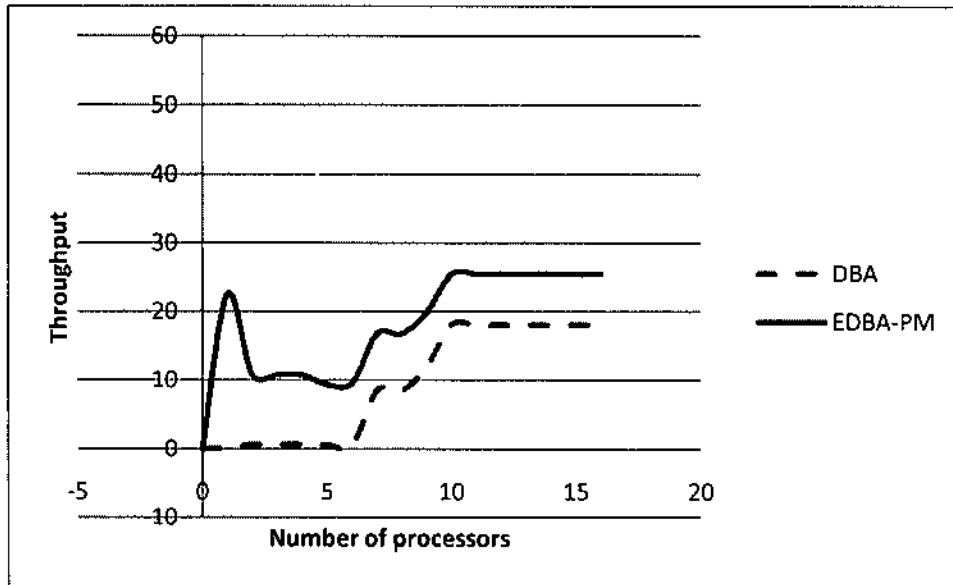


Figure 4.7 Throughput with varying number of processors for EDBA-PM with constant arrival rate of tasks in homogeneous DSs.

4.5.1.2 Variable Arrival Rate of Tasks

In this simulation case, the task arrival rate is not fixed, means that the number of the arrived tasks are not known, and the time between two consecutive tasks changes variably. So the number of tasks that will arrive to the system is larger in this case. As shown in Figure 3.1, which shows the underlying Distributed System structure, there are three task generators, we choose a variable generating rate for all of them; each one can generate a number of tasks that increases exponentially.

In this simulation case each node receives 300 tasks in average and the total number of tasks is 20000tasks. The number of processors used to simulate this case is 16 processors.

4.5.1.2.1 EDBA-CPU-U

When evaluating the algorithms, we notice that the EDBA-CPU-U enhances the performance of the system very well in terms of response time; this result is given in Figure4.8, and 4.9.

The result that is shown in the previous figures is expected because the EDBA-CPU-U behaves as in the system when the arrival rate of tasks is constant. It increases the system performance by 62.9% compared with DBA in varying number of tasks case, and by 49.57% in varying number of processors case.

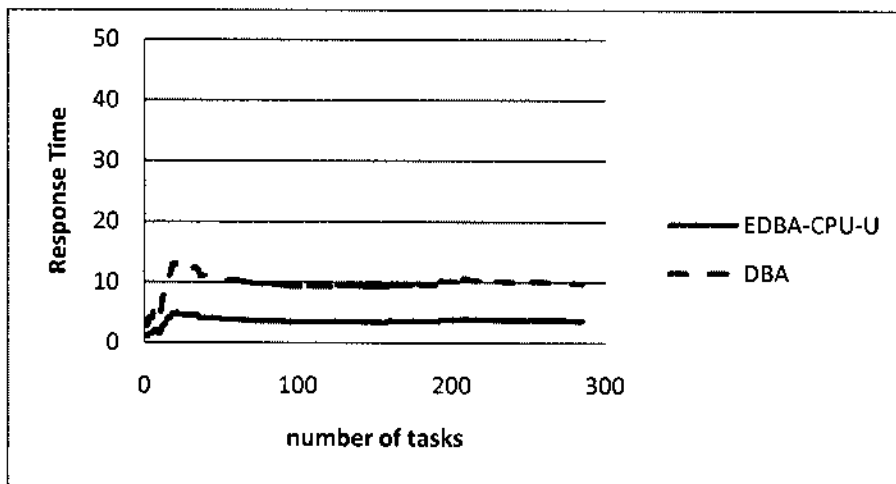


Figure 4.8 Response time with varying number of tasks for EDBA-CPU-U with variable arrival rate of tasks in homogeneous DSs.

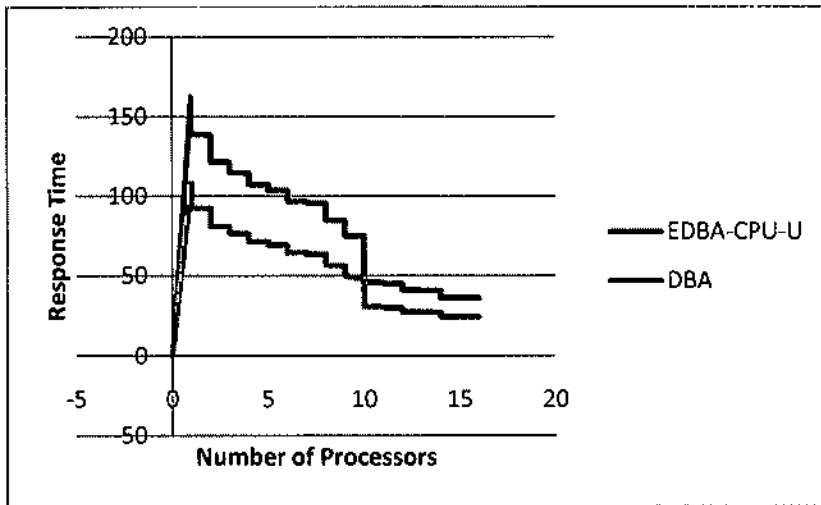


Figure 4.9 Response time with varying number of processors for EDBA-CPU-U with variable arrival rate of tasks in homogeneous DSs.

4.5.1.2.2 EDBA-PM

The proposed algorithm enhances the throughput of the system very well, especially when it is heavily loaded. It uses a threshold that changes adaptively according to the system conditions in order to enhance the performance, this algorithm helps in balancing the load by moving some tasks from the heavily loaded nodes to the lightly loaded nodes. Figure 4.10 and 4.11 shows the result.

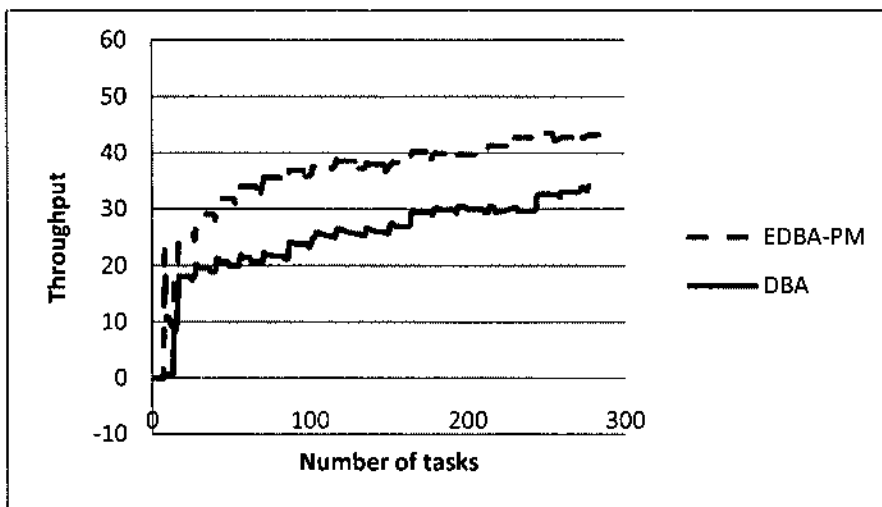


Figure 4.10 Throughput with varying number of tasks for EDBA-PM with constant arrival rate of tasks in homogeneous DSs.

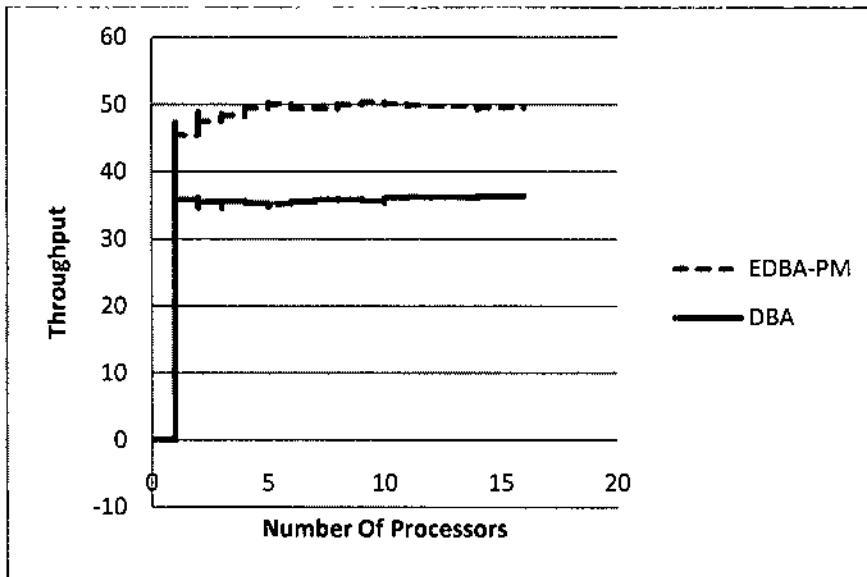


Figure 4.11 Throughput with varying number of tasks for EDDBA-PM with variable arrival rate of tasks in homogeneous DSs.

As the previous figures illustrate the number of processed tasks increases when using EDDBA-PM, because this algorithm helps in distributing tasks among nodes evenly, and achieving a better performance by transferring the tasks from the busy nodes to the idle or lightly loaded nodes and enforce them to execute more tasks.

The EDDBA-PM increases the system throughput by 17.2% with varying number of tasks case and increases the same metric by 22.89% with varying number of processors case.

4.5.2. Heterogeneous Distributed Systems

All nodes in all groups have different service rate and different memory capacity. The communications links that communicate nodes and groups have some delay. The simulation time is set to one hour. Two cases are considered as in the homogeneous system, the first case have constant arrival rate of tasks, and the other case with variable arrival rate of tasks.

4.5.2.1. Constant Arrival Rate of Tasks

In this simulation case each node will receive 280 tasks in average, and the total number of tasks is 18000 tasks. The total number of processors used to simulate the algorithm is 16 processors. The service rate of nodes ranges from 25 bits/second to 300 bit/second.

4.5.2.1.1 EDDBA-CPU-U

Figure 4.12 and 4.13 depicts the performance of the EDDBA-CPU-U when the arrival rate of tasks is constant. In the figure the performance of the two algorithms is simulated with varying number of tasks, and in the second graph the performance is considered when having varying number of processors.

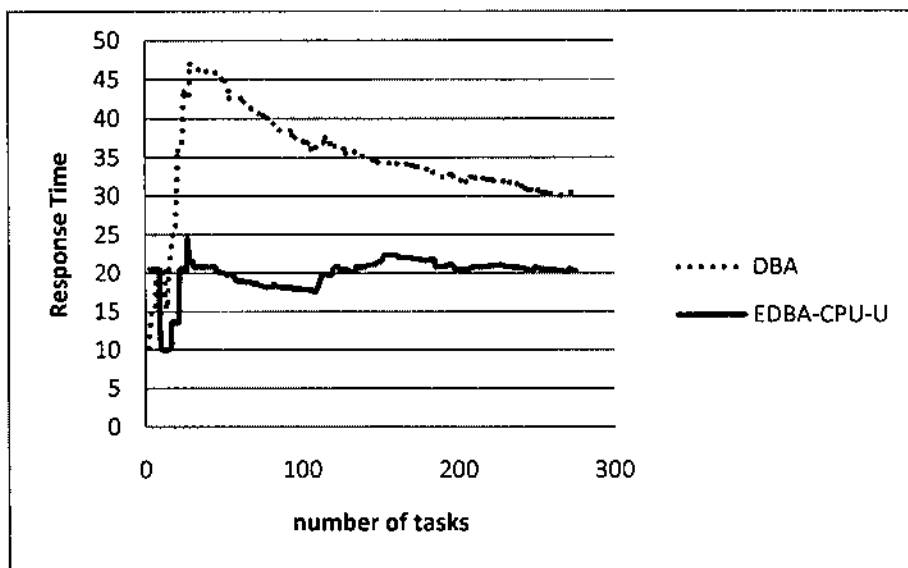


Figure 4.12 Response Time with varying number of tasks for EDDBA-CPU-U with constant arrival rate of tasks in heterogeneous DS

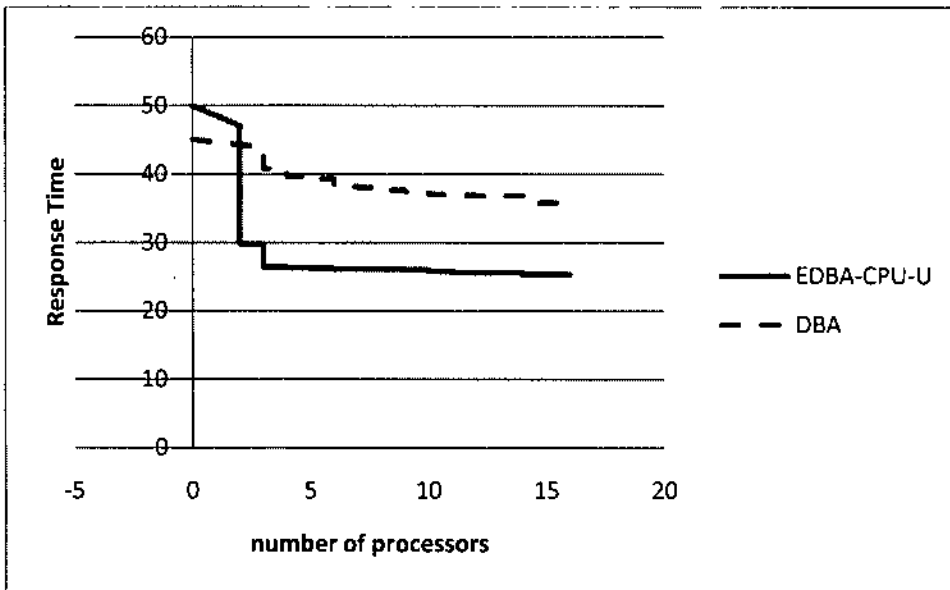


Figure 4.13 Response Time with varying number of procesors for EDDBA-CPU-U with constant arrival rate of tasks in heterogeneous DS

As presented in the previous figure, the EDDBA-CPU-U achieves better performance if response time metric is considered. This result is expected and we can conclude that the proposed algorithm can work properly with both homogeneous and heterogeneous distributed systems.

EDDBA-CPU-U enhances the performance of the system by 34.6% with varying number of tasks, and by 29.49% with varying number of processors.

4.5.2.1.2 EDDBA-PM

The next two figures 4.14 and 4.15 compares between the EDDBA-CPU-U and DBA algorithms in terms of throughput factors.

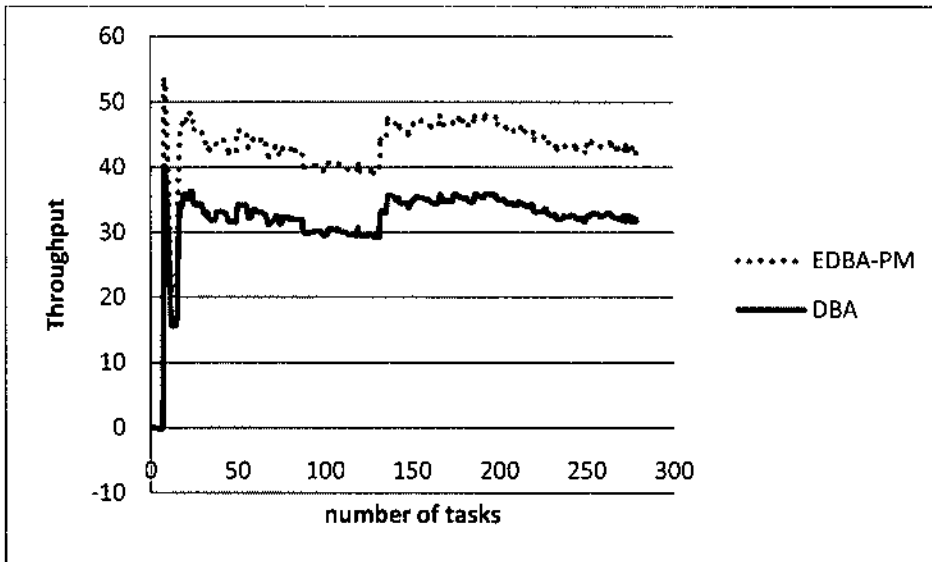


Figure 4.14 Throughput with varying number of tasks for EDDBA-PM with constant arrival rate of tasks in heterogeneous DS

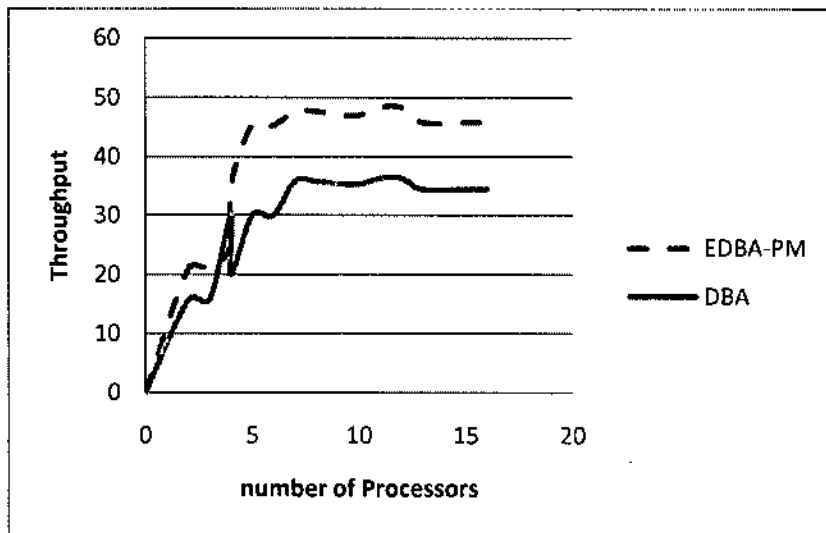


Figure 4.15 Throughput with varying number of processors for EDDBA-PM with constant arrival rate of tasks in heterogeneous DS

The EDDBA-PM performs as in the homogeneous system. It improves the system performance by increasing the number of processed tasks in the system. The percentage difference between the two algorithms is 28.5% when the case is having varying number of tasks and 30.02% with varying number of processors.

4.5.2.2. Variable Arrival Rate of Tasks

Each node in this case receives 300 tasks in average and the total number of tasks is 32000 tasks. The total number of processors is 16.

4.5.2.2.1 EDDBA-CPU-U

Figure 4.16 and 4.17 evaluate the performance of the EDDBA-CPU-U and DBA algorithm in terms of response time factor.

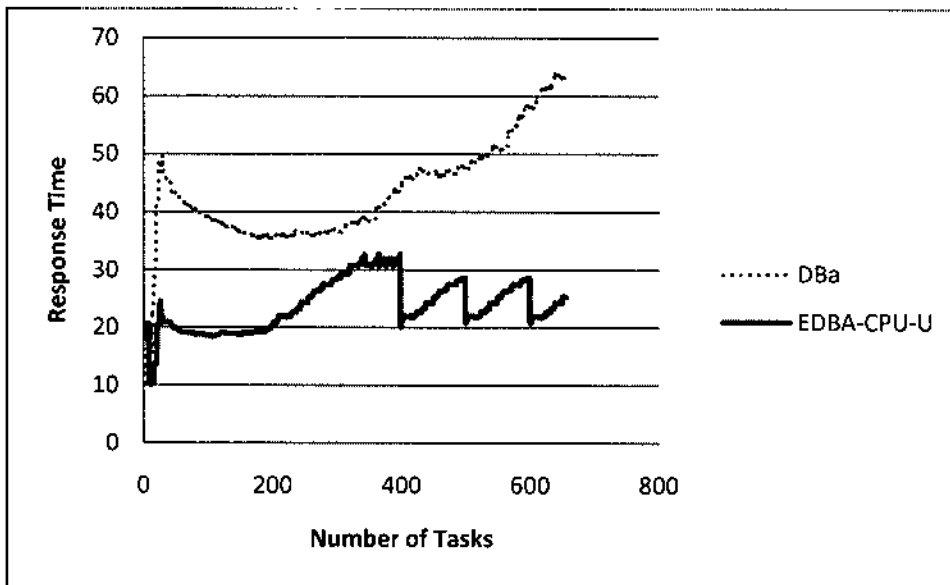


Figure 4.16 Response time with varying number of processors for EDDBA-CPU-U with variable arrival rate of tasks in heterogeneous DS

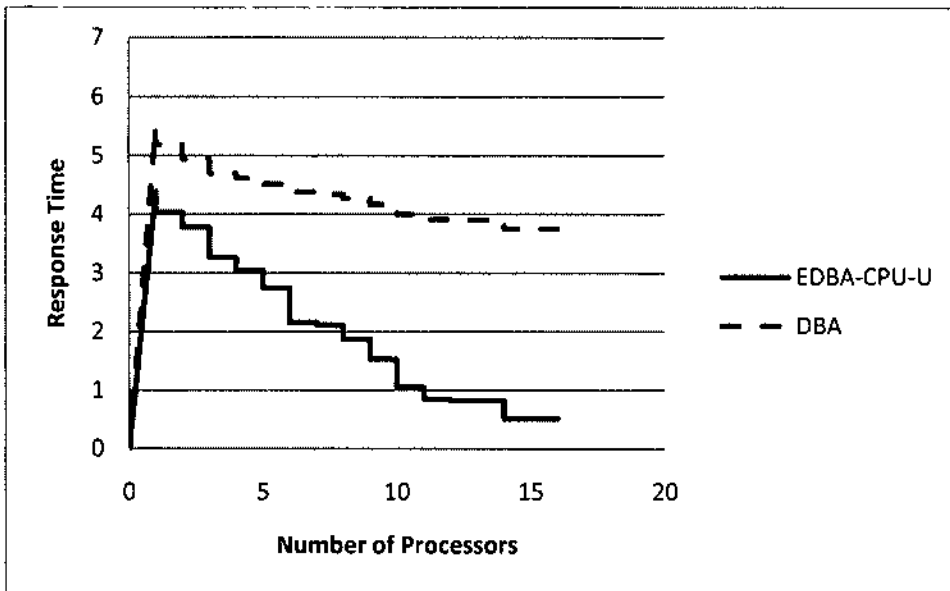


Figure 4.17 Response time with varying number of processors for EDDBA-CPU-U with variable arrival rate of tasks in heterogeneous DS

The response time is smaller in EDDBA-CPU than in the DBA, which gives an efficient system performance. It performs as in the homogeneous system and enhances the system performance in both cases.

In the first comparison when having variable number of tasks the EDDBA-CPU-U enhances the performance of the system by 32% approximately and by 36% in the second case.

4.5.2.2.2 EDDBA-PM

Figure 4.18 and 4.19 shows the throughput metric of the DBA and the EDDBA-PM; we can notice that the enhanced scheme gives a better result than the original scheme. It increases the performance by 34.24% in the first case and by 37% in the second case.

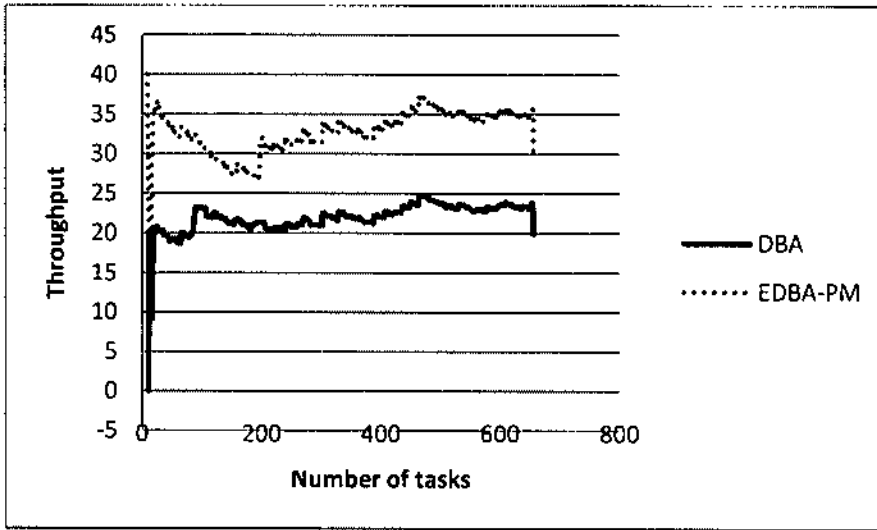


Figure 4.18 Throughput with varying number of tasks for EDDBA-PM with variable arrival rate of tasks in heterogeneous DS

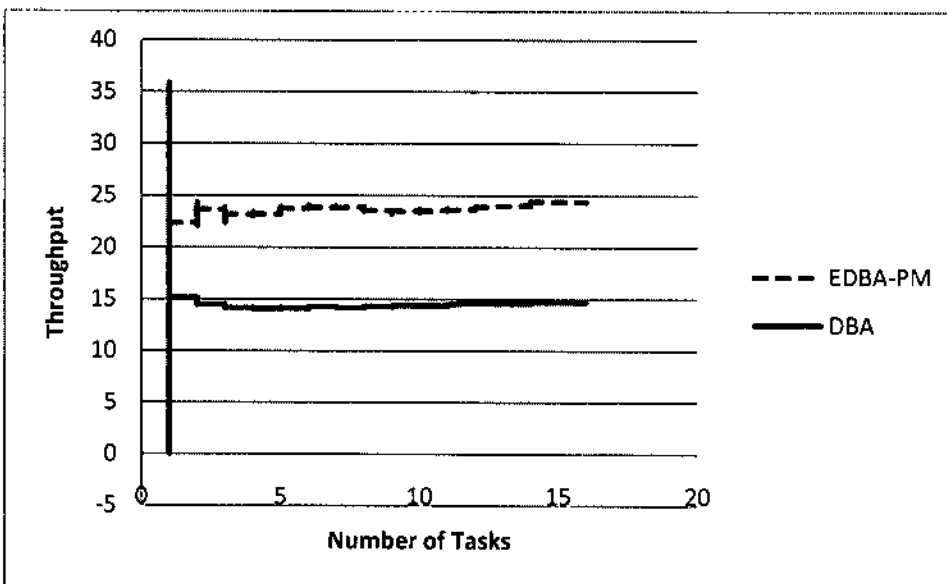


Figure 4.19 Throughput with varying number of processors for EDDBA-PM with variable arrival rate of tasks in heterogeneous DS

4.6 Results Summary

In this section we calculate the percentage difference between the proposed schemes and the original algorithm to conclude how much it improves the system performance.

In homogeneous system with constant arrival rate of tasks the EDDBA-CPU-U decreases the response time by 45% with varying number of tasks, and by 31.07% with varying number of processors compared by DBA. And with variable arrival rate of tasks it decreases the response time by 62.9% with varying number of tasks, and by 49.57% with varying number of processors.

In heterogeneous system with constant arrival rate of tasks the EDDBA-CPU-U decreases the response time by 34.6% with varying number of tasks, and by 29.4% with varying number of processors. In variable arrival rate of tasks it decreases the response time by 32% with varying number of tasks, and by 36% with varying number of processor.

The EDDBA-PM increases the throughput by 15.5% with varying number of tasks in homogeneous system with variable arrival rate of tasks, and by 24.85% with varying number of processors compared by DBA. In variable arrival rate of tasks it increases the throughput by 22.89% with varying number of tasks, and increases the throughput by 17.2% with varying number of processors.

The EDDBA-PM increases the throughput by 28.5% with varying number of tasks in homogeneous system with variable arrival rate of tasks, and by 30.02% with varying number of processors compared by DBA. In variable arrival rate of tasks it increases the throughput by 34.24% with varying number of tasks, and increases the throughput by 37% with varying number of processors.

5. Conclusions and Future Works

5.1. Conclusions

In this work, we have proposed two new schemes EDBA-CPU-U, and EDBA-PM where we aim in it to improve the system performance by increasing the throughput, and at same time decreasing the response time.

In the EDBA-CPU-U a new load index is used to calculate biases that are used to distribute the load among nodes. This index is the CPU utilization rate, which increases the system performance because it helps in decreasing the idling time of nodes and, makes them busy in most of the time. As a result the nodes will process more tasks and the throughput will increase.

In the EDBA-PM the process migration policy is used to enhance the system performance by transferring some load from the nodes that is not able to execute the tasks to another node. The source node, the destination node, and the transferred task are determined according to some policy. According to this policy, nodes will have the same load state approximately, and increasing the system performance as a result.

The two proposed schemes are evaluated with twelve cases in order to achieve our goal. In homogeneous system with constant arrival rate of tasks the EDBA-CPU-U decreases the response time by 45% with varying number of tasks, and by 31.07% with varying number of processors compared by DBA. And with variable arrival rate of tasks it decreases the response time by 62.9% with varying number of tasks, and by 49.57% with varying number of processors.

In heterogeneous system with constant arrival rate of tasks the EDBA-CPU-U decreases the response time by 34.6% with varying number of tasks, and by 29.4% with varying

number of processors. In variable arrival rate of tasks it decreases the response time by 32% with varying number of tasks, and by 36% with varying number of processor.

The EDDBA-PM increases the throughput by 15.5% with varying number of tasks in homogeneous system with variable arrival rate of tasks, and by 24.85% with varying number of processors compared by DBA. In variable arrival rate of tasks it increases the throughput by 22.89% with varying number of tasks, and increases the throughput by 17.2% with varying number of processors.

The EDDBA-PM increases the throughput by 28.5% with varying number of tasks in homogeneous system with variable arrival rate of tasks, and by 30.02% with varying number of processors compared by DBA. In variable arrival rate of tasks it increases the throughput by 34.24% with varying number of tasks, and increases the throughput by 37% with varying number of processors.

5.2 Future Works

In this study all tasks that generated by the client and arrived to the nodes have the same size, different tasks sizes could be taken onto consideration in any future investigation. Different sizes of tasks may affect the performance of the system, so future work may consider this investigation to evaluate how different sizes of jobs affect the system. The comparison of our methods with other new methods could also be considered as a future work.

REFERENCES

- Ahn Hyo Cheol, Youn Hee Yong, Jeon Kyu Yeong, and Lee Kyu Seol, (2007), Dynamic Load Balancing for Large-scale Distributed System with Intelligent Fuzzy Controller, **IEEE International Conference**, pp. 576 – 581,
- Akhtar Muhammad Waseem, and Kechadi M-Tahar, (2006), On the Efficiency of Dynamic Load Balancing on P2P Irregular Network Topologies, Proceedings of The **Fifth International Symposium on Parallel and Distributed Computing (ISPDC'06)**, 0-7695-2638-1/06, **IEEE**.
- Ammar H, and Deng Su, (1988), A Simple Dynamic Load Balancing Algorithm For Homogeneous Distributed Systems, **ACM Annual Computer Science Conference, Proceedings of the 1988 ACM sixteenth annual conference on Computer science**, pp. Pages: 314 – 319, ACM 0-89791-260-8/88/0002/0314.
- Ioana Banicescu, Florina M. Ciorba, and Ricolindo L. Cariño, 2009, Towards the robustness of dynamic loop scheduling on large-scale heterogeneous distributed systems, 2009 **Eighth International Symposium on Parallel and Distributed Computing**, 978-0-7695-3680-4/09 **IEEE**, pp.129-132.
- Barazandeh Iman, and Mortazavi Seyed Saeedolah,(2009/a), Two Hierarchical Dynamic Load Balancing Algorithms in Distributed Systems, 2009 **Second International Conference on Computer and Electrical Engineering**, 978-0-7695-3925-6/09, **IEEE**, PP.516-521.

Iman, Mortazavi Seyed Saeedolah, and Mortazavi Amir Masoud, (2009/b), Intelligent Fuzzy based Biasing Load Balancing Algorithm in Distributed Systems, **Proceedings of the 2009 IEEE 9th Malaysia International Conference on Communications**.

J. Barbosa, and Moreira Belmiro,(2009), Dynamic job scheduling on heterogeneous clusters, **Eighth International Symposium on Parallel and Distributed Computing**, 978-0-7695-3680-4/2009, **IEEE**, pp.3-10.

Mahieddine K. Benmohammed, and Dew P. M., (1994), a periodic symmetrically-initiated load balancing algorithm for distributed systems, **Distributed Computing Systems, Proceedings of the 14th International Conference, IEEE**, pp.616 – 623.

Broberg James, Tari Zahir, and Zeephongsekul Panlop, (2005), **Principles of Distributed Systems Book**, Task assignment based on prioritizing traffic flows Chapter, Publisher: Springer Berlin / Heidelberg, DOI: 10.1007/11516798_30, Vol.3544/2005, Book Series: Lecture Notes in Computer Science, pp.415-430.ft

Broberg James, Tari Zahir, and Zeephongsekul Panlop, (2006), Task assignment with work-conserving migration, **parallel Computing** 32 (2006), pp.808–830, **Science Direct**, journal website: www.elsevier.com/locate/parco.

Campos Luis Miguel, and Isaac Scherson D, (2000), Rate of change load balancing in distributed and parallel systems, **Science Direct, Parallel Computing**, 26 (2000), pp.1213-1230.

Cao Jiannong, Bennett Graeme , and Zhang Kang, 2000, Direct execution simulation of load balancing algorithms with real workload distribution, **The Journal of Systems and Software**, **Science Direct**, www.elsevier.com/locate/jss .

Chhabra Amit, Singh Gurvinder, Waraich Sandeep Singh, Sidhu Bhavneet, and Kumar Gaurav, (2006), Qualitative Parametric Comparison of Load Balancing Algorithms in Parallel and Distributed Computing Environment, **World Academy of Science, Engineering and Technology**, pp 39-42.

Chen Jong-Chen, Liao Guo-Xun, Hsie Jr-Sung, and Liao Cheng-Hua, (2008), A study of the contribution made by evolutionary learning on dynamic load-balancing problems in distributed computing systems, **Science Direct, Expert Systems with Applications** 34 (2008), www.elsevier.com/locate/eswa.

Choi Eunmi, (2004), Performance test and analysis for an adaptive load balancing mechanism on distributed server cluster systems, **Future Generation Computer Systems** 20 (2004), **Science Direct**, www.elsevier.com/locate/future .

Devine Karen D, Boman Erik G., Heaphy Robert T, Hendrickson Bruce A, Teresco James D, Faik Jamal, Flaherty Joseph E, and Gervasioc Luis G., (2005), New challenges in dynamic load balancing, **Applied Numerical Mathematics** 52 (2005), **Science Direct**, www.elsevier.com/locate/apnum.

Fedorov Andriy, and Chrisochoides Nikos, (2004), Communication Support for Dynamic Load Balancing of Irregular Adaptive Applications, **International Conference on Parallel Processing Workshops, IEEE**, pp. 555 - 562 .

Fukuda Kensuke, Hirotsu Toshio, Kurihara Satoshi, Akashi Osamu, Sato Shin-ya, and Sugawara Toshiharu, (2007), **Emergent Intelligence of Networked Agents Book**, The Impact of Network Model on Performance of load balancing Chapter, Publisher: Springer Berlin / Heidelberg, PP. 23-37, Volume 56/2007, Studies in Computational Intelligence Book series.

Ghosh Bhaskar, and Muthukrishnan S., (1994), Dynamic Load Balancing in Parallel and Distributed Networks by Random Matchings, **ACM Symposium on Parallel Algorithms and Architectures, Proceedings of the sixth annual ACM symposium on Parallel algorithms and architectures**, pp. 226 – 235.

Grosu Daniel, and Chronopoulos Anthony T., (2005), Non-cooperative load balancing in distributed systems, **Journal of parallel and Distributed computing**.65 (2005) 1022-1034, **Science Direct**, www.elsevier.com/locate/ejor.

Hac Anna, (1989), Load balancing in distributed systems: a summary, **ACM SIGMETRICS Performance Evaluation Review**, Vol.16, PP.17 -19.

BALTER MOR HARCHOL-, and DOWNEY ALLEN B., 1997, Exploiting Process Lifetime Distributions for Dynamic Load Balancing, **ACM Transactions on Computer Systems**, Vol. 15, No. 3, August 1997, Pages 253–285.

Chang Huang Ming, Hosseini S. Hossein, and Vairavan K., (2003), A Receiver-Initiated Load Balancing Method In Computer Networks Using Fuzzy Logic Control, **Global Telecommunications Conference, 2003. GLOBECOM '03, IEEE**, pp.4028 – 4033, Vol.7.

Jain Parveen, and Gupta Daya, (2009), An Algorithm for Dynamic Load Balancing in Distributed Systems with Multiple Supporting Nodes by Exploiting the Interrupt Service, **International Journal of Recent Trends in Engineering**, Vol 1, No. 1, May 2009, pp. 232- 236.

Kwok Yu-Kwong, and Cheung Lap-Sun, (2004), A new fuzzy-decision based load balancing system for distributed object computing, **journal of parallel and distributed computing**, 64 (2004) 238–253, **Science Direct**, www.elsevier.com/locate/jpdc.

Larroca Federico, and Rougier Jean-Louis, (2009), **Traffic Management and Traffic Engineering for the Future Internet Book**, A Fair and Dynamic Load-Balancing Mechanism chapter, Publisher: Springer Berlin / Heidelberg, pp.36-52, DOI 10.1007/978-3-642-04576-9_3, Volume 5464/2009, Lecture Notes in Computer Science book series.

Li Keqin, (1998), Deterministic and Randomized Algorithms for Distributed On-line Task Assignment and Load Balancing without Load Status Information, **ACM symposium on Applied Computing**, 0-89791-969-6/98/0002, pp.613-622.

Li Keqin, (2002), Minimizing the Probability of Load Imbalance in Heterogeneous Distributed Computing Systems, **journal of mathematical and computer modeling** 36 (2002)1075-1084, **Science Direct**, journal website: www.elsevier.com/locate/eswa.

Li Yajun, Yang Yuhang, Ma Maode, and Liang Zhou, (2009), A hybrid load balancing strategy of sequential tasks for grid computing environments, **Future Generation Computer Systems** 25 (2009) 819-828, **Science Direct**.

Nouri Setareh, and Parsa Saeed, 2009, A Non-cooperative Approach For Load Balancing In Heterogeneous Distributed, 2009 **Fourth International Conference on Computer Sciences and Convergence Information Technology Computing Platform**, 978-0-7695-3896-9/09, **IEEE**, PP.756-761.

Penmatsa, and Chronopoulos Anthony T., (2007), Dynamic Multi-User Load Balancing in Distributed Systems, 1-4244-0910-1/07, **Parallel and Distributed Processing Symposium, 2007, IPDPS 2007. IEEE International**, pp.1-10.

Perez Christian, (1997), Load Balancing HPF Programs by Migrating Virtual Processors, **IEEE**.

Razzaque Md. Abdur, and Hong Choong Seon, (2007), Dynamic Load Balancing in Distributed System: An Efficient Approach, **JCCI conference 2007**, Phoenix Park, South Korea, May 2-4, 2007, **Korea Research Foundation Grant**, (KRF-2006-521-D00394).

ROSS KEITH W., and YAO DAVID D., (1991), Optimal Load Balancing and Scheduling in a Distributed Computer System, **Journal of Association Computing Machinery**, Vol 38, No.3, July1991, pp 676-690.

Savvas Ilias K, and Kechadi M-Tahar, (2004/a), Dynamic Task Scheduling in Computing Cluster Environments, **Parallel and Distributed Computing, IEEE**, Proceedings of the ISPDC/HeteroPar'04, 0-7695-2210-6/04,pp.372 – 379, July/2004.

Savvas Ilias K., and Kechadi M-Tahar, (2004/b),Performance study of a dynamic task scheduling for heterogeneous distributed systems, **Operational Research journal**, Vol.4, No.3/September 2004, PP.291-303, Springer Berlin / Heidelberg.

Ilias K.Savvas, and M-Tahar Kechadi, (2007), Efficient Load Balancing on Irregular Network Topologies Using B+tree Structures, **Sixth International Symposium on Parallel and Distributed Computing (ISPDC'07)**, 0-7695-2936-4/07, IEEE.

Sharma Sandeep, Singh Sarabjit, and Sharma Meenakshi, (2008), Performance Analysis of Load Balancing Algorithms, **World Academy of Science, Engineering and Technology** ,38 (2008), pp. 269- 272.

Spies Fraquois, (1996), Modeling of optimal load balancing strategy using queuing theory, **Micro-processing and Microprogramming**, Vol.41, pp.555-570, **Science Direct**.

Vaughan John G, (1995), Incorporating job sizes in distributed load balancing, **Micro-processing and Microprogramming** 41 (1995), pp.111-119, **Science Direct**.

Wang Jun, Chen Jian-Wen, Yong-Liang Wang, and Zheng Di, 2009, Intelligent Load Balancing Strategies for Complex Distributed Simulation Applications, **2009 International Conference on Computational Intelligence and Security**, 978-0-7695-3931-7/09, **IEEE**, PP.182-186.

Watts Jerrell, Rieffel Marc, and Taylor Stephen, (1996), Parallel Algorithms for Irregularly Structured Problems Book, Practical dynamic load balancing for irregular problems chapter, Publisher : Springer Berlin / Heidelberg, Volume 1117/1996, PP: 299-306.

Xu Jian, and Hwang Kai, (1990), Heuristic Methods for Dynamic Load Balancing in A Message-Passing Supercomputer, **Conference on High Performance Networking and Computing, Proceedings of the 1990 ACM/IEEE conference on Supercomputing**, **IEEE Computer Society Press** , Pages: 888 – 897.

Yan K.Q., Wang S.C., Chang C.P., and Lin J.S., (2007), A hybrid load balancing policy underlying grid computing environment, **Computer Standards & Interfaces** 29 (2007) 161–173, **Science Direct**, www.elsevier.com/locate/ejor.

تقييم أداء الخوارزميات الديناميكية لتحميل الموازنة في نظم الحوسبة الموزعة

إعداد

رانيه علي ناجي عتوم

المشرف

الدكتور سامي السرحان

ملخص

في هذه الدراسة نركز على تحقيق الموازنة في نظم توزيع ذات الهيكل الهرمي من خلال تعزيز خوارزمية الحيوية. في الخوارزمية الأصلية ويقاس حالة التحميل من خلال طول الانتظار وحدة المعالجة المركزية الحالية لتوزيع الحمل بين العقد على أساس حالة تحميل الحالي.

وسيقدم اثنان من الخوارزميات المقترحة ، في الخوارزمية الأولى هي قياس حالة حمل من طول قائمة انتظار وحدة المعالجة المركزية كما هو الحال في الخوارزمية الأصلية ، بالإضافة إلى معدل استخدام وحدة المعالجة المركزية. تحميل هذا المؤشر الجديد (معدل استخدام وحدة المعالجة المركزية) يزيد من أداء النظام لأنه يجعل العقد مشغول معظم الوقت. لأنه يزيد من أداء النظام بمقدار 62.9 ٪ من حيث عامل استجابة الوقت.

وفي الخوارزمية الثانية المقترحة تم إدراج سياسات هجرة الوظيفة بين العقد. والقيمة المحددة التي تتغير بنكيف تتم مقارنتها مع حمولة العقد بشكل دوري ، إذا كانت العقدة هي محملة بشكل كبير (حالاته تحميل أكبر من القيمة) ، يتم تنفيذ سياسة الهجرة العملية عن طريق نقل بعض من حمولتها إلى عقدة أخرى.

هذا يزيد من تعزيز أداء النظام من حيث الإنتاجية وخصوصا عندما يتم تحميل النظام بشكل كبير. سياسة الهجرة يساعد في تحقيق موازنة التحميل لأنه يجعل من كل العقد لها نفس التحميل تقريبا ، وزيادة أداء النظام وفقا لذلك. لأنه يزيد من أداء النظام من حيث عامل الإنتاجية بنسبة 37 %.

684770