Scheduling Real-time Divisible Loads in Cluster Computing Environment

A thesis submitted in partial fulfillment
of the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Abhinandan Aryadipta
Avinash Behera

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India

May 2012

Scheduling Real-time Divisible Loads in Cluster Computing Environment

A thesis submitted in partial fulfillment
of the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Abhinandan Aryadipta
Avinash Behera

under the guidance of

Bibhudatta Sahoo

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India

May 2012

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India.

Certificate

This is to certify that the work in the thesis entitled *Scheduling of Real-time Divisible Loads in Cluster Computing Environment* submitted by *Abhinandan Aryadipta* and *Avinash Behera* in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering during the session 2011-2012 in the department of Computer Science and Engineering, National Institute of Technology Rourkela (Deemed University) is an authentic work carried out under my supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other University/Institute for the award of any Degree or Diploma.

Bibhudatta Sahoo
Assistant Professor
Dept. of Computer Science & Engineering
National Institute of Technology
Rourkela-769008, Odisha (India)

Place: NIT Rourkela
Date: 14 May 2012

Acknowledgement

No thesis is the result of the effort of one individual. It is a product resulting from the synthesis of the valuable contributions of many people. We are deeply grateful to our thesis supervisor, Bibhudatta Sahoo, Assistant Professor, Department of CSE, for his guidance, support, motivation and encouragement throughout the period this work was carried out. His readiness for consultation at all times, his educative comments, his concern and assistance even with practical things have been invaluable.

We would also like to thank all professors and lecturers, and members of the Department of Computer Science and Engineering, for their generous help in various ways for the completion of this thesis. We are extremely thankful to all our fellow students for their friendly co-operation.

Last but not the least, we would like to thank our parents for their continued support and encouragement at all times.

Abhinandan Aryadipta
Roll No. 108CS010
NIT Rourkela

Avinash Behera
Roll No. 108CS009
NIT Rourkela

Abstract

The significance of cluster computing in solving massively parallel workloads is tremendous. Divisible Load Theory has proven to be very successful in optimizing the usage of the system resources by partitioning the arbitrarily divisible loads adequately among the cluster nodes. Arbitrarily divisible loads have significant real-world applications in high energy and particle physics. In this thesis, various algorithms for a cluster computing environment are studied including the ones dealing with divisible load theory confirming DLT based algorithms performing better in most cases. The loads that are considered in this thesis are hard real-time tasks with associated deadlines. Specifically, a comparison is made between clusters with one where the head node doesn't participate in processing of the work-loads with the other where the head node does participate in processing of the work-loads. A new mathematical formula is derived for the task execution time corresponding to the new scenario of head node possessing front-end processing capability. The existing algorithms corresponding to Real-Time Divisible Load Theory are then implemented using this new formula to examine the scheduling performance in this new scenario compared to the conventional scenario where the head node lacks front-end processing capability.

Contents

List of Figures

List of Tables

List of Abbreviations

RT-DLT : Real-Time Divisible Load Theory

DLT : Divisible Load Theory

EDF : Earliest Deadline First

FIFO : First In First Out

MWF : Maximum Workload derivative First

OPR : Optimal Partitioning Rule

EPR : Equal Partitioning Rule

AN : All Nodes

MN : Minimum number of Nodes

CPU : Central Processing Unit

CMS : Compact Moon Solenoid

LHC : Large Hadron Collider

ATLAS : AToroidal LHC Apparatus

CERN : European Laboratory for Particle Physics

IEEE : Institute of Electrical and Electrical Engineers

TRR : Task-Rejection-Ratio

List of Symbols

lamdainv	: Mean of the inter-arrival time of the tasks
N	: Number of processing nodes in the cluster
n	: Number of processing nodes assigned to a task
A	: Arrival time of a task
A_i	: Arrival Time of the i^{th} task
σ	: Data Size of a task
σ_i	: Data Size of the i^{th} task
D	: Relative Deadline of a task
D_i	: Relative Deadline of the i^{th} task
P_0	: Head Node of the cluster
P_i	: i^{th} processing node of the cluster
Cps	: Cost of executing a unit workload on a single processing node
Cms	: Cost of transmitting a unit workload to a single processing node
$Cp(\sigma)$	: Cost function giving the cost of executing a workload of size σ on a single processing node
$Cm(\sigma)$	: Cost function giving the cost of transmitting a workload of size σ on a single processing node
$\Sigma(\sigma,n)$	: Execution time function which gives the time taken to complete by a task having data size σ and to which n processing nodes have been assigned
α_i	: Fraction of the total data size of a task assigned to the i^{th} processing node
Cpi	: Time taken to execute a unit workload on the i^{th} processor
Cmi	: Time taken to transmit a unit workload to the i^{th} processor
r_i	: Release-time of the i^{th} processor
a_i	: Cost of executing a task on the i^{th} processor for unit time
Y	: Yes
N	: No

β	:	Computing Factor = Cps / (Cps + Cms)
avgσ	:	Average load size of the tasks
DCRatio	:	Ratio of avgD to Σ(avgσ, N)
avgD	:	Average relative deadline of the tasks
n^{min}	:	Minimum number of processing nodes needed by a task to complete by its deadline

Chapter 1

Introduction

1.1 Introduction

With the tremendous growth in technology and its applications, there has been equal growth in the problems related to computing. Previously, what a computer could do was very limited compared to the present scenario. With better capabilities, we want to solve greater problems with the help of computers. These greater problems increased as our desire to do everything automatically using computers increased. Ultimately, we now look at problems that are extremely difficult to solve. They are exceedingly computationally intensive and hence, require more than a single computer to be solved. Here is where cluster computing comes into the picture.

These computationally intensive problems that need to be solved today are accompanied by massively parallel workloads [15, 16]. Much research exists on the scheduling of real time loads on a uniprocessor system. But the problem of scheduling tasks when multiple processors are available is a relatively new field. Again, when we look at the nature of tasks, they vary as to the manner in which they could be partitioned. The power of massively parallel workloads has not been fully realized as these are highly suitable for those applications that need massively parallel workloads. The field of parallel execution of such

divisible loads on more than one processor has been encompassed under the umbrella of Divisible Load Theory. Several algorithms have been proposed to schedule the real-time divisible loads on more than one processing node. In this thesis, we have studied some of these algorithms to arrive at results that support the existing findings. Again, one constant factor in the existing body of work dealing with scheduling of real-time divisible loads on clusters is that the head node lacks front-end processing capabilities. The head-node, in all these cases, basically receives the tasks, partitions them and assigns the subtasks to the various processing nodes. In this thesis, we have taken a system where the head node participates in direct computation of the task too apart from performing its usual functions. We have studied how a head node having front-end processing capability affects the overall scheduling performance.

1.2 Clusters

Cluster computing refers to a group of computers working together to solve a bigger problem that cannot be solved using a single standalone computer. A cluster can be as small as tens of computers or as big as thousands of computers or even more. The sole purpose is to harness the power of all these systems together so that computationally intensive tasks can be solved. But then again, with the increase in the size of the cluster, the complexity also increases. And to handle such complexity, proper resource management is essential. And with proper resource management, not only does a cluster handle enormous tasks with ease, but also provides major cost benefits.

1.2.1 Classification of Clusters

A cluster can be classified according to its functionality as follows:

High Availability Clusters

These clusters are primarily aimed at providing availability of the clusters and ensuring that the cluster based applications run round the clock. This happens successfully when at the time of failure of a node of a cluster, the applications running on that node failover to another node. In other words, the application runs even if the node it is running on fails. During repairs or maintenance of the nodes, the availability of the cluster is not affected. Besides, if a node fails, it can be repaired and added back to the cluster.

Load Balancing Clusters

Here, the load is balanced among the cluster nodes for requests for same content. Each node knows how to handle the request for the same application. In case of a node failure, the load for that node is redistributed among the remaining nodes that are available. Each node processes the task as a whole and the task is not further divided in case of load balancing clusters.

Distributed Processing Clusters

These are CPU intensive clusters which in contrast to load balancing and high availability clusters, divide the tasks into sub tasks which can be executed in parallel on different cluster nodes. These clusters are particularly useful for data intensive tasks. The major applications include scientific analysis in fields like high energy and particle physics and financial data analysis.

1.2.2 Benefits of Cluster Computing

The major benefits of using a cluster instead of a stand-alone system can be summarized as follows:

Scalability

Any modification to a stand-alone system needs a reconfiguration of the entire system. Increasing or decreasing the performance to balance with the cost is very difficult in case of these systems. But in a cluster, adding or removing nodes is much easier and thus the performance of the system can be scaled up or down as per the requirement.

Scavenging

Any system's actual utilization of the resources is a mere ten per cent of the total resources. And this per cent utilization decreases further as the complexity of the system grows. Hence, it is suitable to have a number of nodes working together to form a system rather than having a stand-alone system.

Performance

A cluster has significant performance improvement over a stand-alone system of the same cost. This is due to ease of availability and low cost of the nodes that form a cluster as compared to the more complex stand-alone system which is costlier. But proper resource management is also essential to get the maximum performance benefits.

Availability

Any failure in a stand-alone system would lead to the failure of the entire system. This is in contrast to the clusters where the failure of any node would not lead to the failure of the entire system. Besides, in a cluster the node that has failed can be replaced or repaired to restore the initial performance of the cluster. In other words, the cluster remains always available to the applications running on it at an abstract level.

Cost

With the growing decrease in cost of the average performing computer systems, clusters can be easily made at a much cheaper cost keeping the system much more simple while in stand-alone systems the cost for acquiring the same performance is significantly higher. Even the maintenance costs are much cheaper in case of the clusters.

1.3 Divisible Loads

1.3.1 Real-time Divisible Loads

The computational loads can be divided into two categories: indivisible and divisible. Indivisible loads are those that cannot be further divided and hence must be assigned to a single processor. These are essentially sequential jobs. On the other hand, a divisible load is one which can be further divided into a number of sub-loads and these sub-loads can be executed in parallel on different processors. The divisible loads can be further divided into two categories: modularly divisible and arbitrarily divisible loads. Modularly divisible loads are those which can be divided into modules, each having a fixed amount of load and are often described by a task (or processing) graph. Arbitrarily divisible loads are those that can be divided arbitrarily into a number of fractions and can be executed in parallel without any task graph relation.

Divisible load theory is used in order to partition the tasks so that they complete in optimal time duration and the cost is reduced. Divisible load theory is based on the concept that when the load is divided among multiple nodes, the minimum time the system takes to complete the entire load is realizable only when all the nodes complete processing the sub-loads for the given load at the same time. Using divisible load theory in cluster computing has had significant performance benefits. Lin et al. in [9, 10, 11, 13] applied DLT to real-time load scheduling in clusters and found answers as to how to parti-

tion a task and assign nodes to it optimally. The applications of the arbitrarily divisible loads include high energy and particle physics experiments, Compact Moon Solenoid (CMS) and AToroidal LHC Apparatus (ATLAS) Experiments at the Large Hadron Collider (LHC) at CERN which form the motivation of our thesis.

In this thesis, we consider arbitrarily divisible loads which form the basic block for load partitioning. The loads that we consider are hard real-time loads with associated deadlines. On the basis of deadlines, loads can be categorized as real-time and non-real-time loads. Non real-time loads are those that do not have an associated deadline. Real-time loads on the other hand have associated deadlines. Real-time loads can be further divided into soft real-time loads and hard real-time loads. Hard real-time loads are those that must be completed by their deadlines. If the results are obtained after the deadlines, then they are of absolutely no use and the entire processing would be a wastage of resources and meaningless. Soft real-time loads also have deadlines but they are not strictly bound to them. The utility of the result decreases as the deadline is crossed but the result may have some significance even beyond the deadline.

1.4 Literature Review

The need for implementation of real-time application systems on multiprocessor platforms is arising due to their increasing complexity. But the traditional models that exist for representing real-time workloads do so only for uniprocessor systems. This entails several drawbacks. One of them is the complete ignorance of the parallel execution of tasks which is possible in case of multiprocessor systems. This parallel execution of tasks is going to be a key need of future generations of real-time systems. Some recent research [1, 2, 3, 4, 5, 6, 7] exists on extending these traditional task models to include the possibility of parallel execution of tasks over more than one processor. In [4, 5, 6, 7], the task's specifications explicitly include the rate at which each task would

execute upon assignment to different number of processors.

Applying Divisible Load Theory [DLT] to real-time workloads also provides for the parallel execution of tasks and hence overcomes the deficiency of the traditional models of workload representation [9, 10, 11, 12, 13, 14]. Real-time divisible loads are the divisible loads that have deadlines associated with them and that are useful only when they complete by their associated deadlines. DLT makes the assumption that tasks are parallelizable to an arbitrary degree and that there are overheads associated with the partition of the tasks into subtasks and the distribution of these subtasks among the various processors available. Hence, DLT incorporates massively parallel workloads. Example of such workloads can easily be found in high energy and particle physics. For instance, the CMS (Compact Muon Solenoid) [15] and ATLAS (AToroidal LHC Apparatus) [16] projects, which are associated with the Large Hadron Collider (LHC) at CERN (European Laboratory for Particle Physics), execute cluster based applications with arbitrarily divisible loads. Usually, all elements in such computational loads demand an identical type of processing and relative to the huge total computation, the processing on each individual element is infinitesimally small [9]. [25] also shows that DLT is particularly well suited to the processing of very long linear files such as those that occur in signal and image processing, experimental data processing, Kalman filtering, cryptography and genetic algorithms.

In [9, 10, 11, 13] Lin et al. extended Divisible Load Theory to real-time workloads and found elegant solutions to the following problems:

- Given a divisible job and a specified number of processors, how to partition this job among these processors so that it completes its execution in minimum time.

- Given a divisible real-time job, what will be the minimum number of processors that should be assigned to this job so that it completes its execution by its deadline.

Name of Paper	A	σ	D	Cps	Cms	Cpi	Cmi	r_i	a_i
Lin et al. [11]	Y	Y	Y	Y	Y	N	N	N	N
Lin et al. [12]	Y	Y	Y	N	N	Y	Y	Y	N
Chuprat et al. [19]	Y	Y	Y	Y	Y	N	N	Y	N
Chuprat [17]	Y	Y	Y	N	N	Y	Y	Y	N
Chuprat et al. [20]	Y	Y	Y	N	N	Y	Y	Y	Y

Table 1.1: Comparison of various papers on RT-DLT with respect to the parameters considered

Some research existed on application of DLT to heterogeneous clusters [21, 22, 23], but they didn't deal with real-time workloads. Also it was assumed in the previous works that all the processors are available to a task at the same time. In [12], Lin et al. examined the problem when the processors are available at different times by converting it to a problem of heterogeneous cluster and hence, they modified it to a situation where the processors have identical ready times but different processing capabilities. But this led only to an approximate solution. In [19], Chuprat extended this approach to apply to heterogeneous clusters where links to the different processors have different transmission costs and each processor has different processing capability. Chuprat et al. in [20] extended their work to include another parameter a_i, the cost of executing a task on a processor and this cost is different for different processors. The current work on RT-DLT can thus be summarized in the form of the table 1.1.

The various symbols can be described as follows:

- Y : Yes
- N : No
- A : Arrival time of a task
- σ : Data size of a task

- D : Relative deadline of a task
- Cps : Time taken to execute a unit workload on a single processor
- Cms : Time taken to transmit a unit workload to a processor
- Cpi : Time taken to execute a unit workload on the i^{th} processor
- Cmi : Time taken to transmit a unit workload to the i^{th} processor
- r_i : Release-time of the i^{th} processor
- a_i : Cost of executing a task on the i^{th} processor for unit time

1.5 Motivation

The present literature on scheduling of Real-Time Divisible Loads on clusters does not take into account the possibility of the head node taking part in actual computation of the tasks. Hence, we have implemented the existing algorithms to examine this particular scenario and obtained results demonstrating the behavior of existing scheduling algorithm in such a scenario.

1.6 Problem Statement

The problem that we have investigated is of scheduling arbitrary divisible loads of the form (A,σ,D) on a homogenous cluster (N,Cms,Cps) such that both the Task-Rejection-Ratio and the Total Execution Time are minimized. The cluster in this case has the specific property that the head node possesses front-end processing capability. We have also studied the existing algorithms to confirm that DLT is beneficial in the scheduling of real-time workloads in clusters.

1.7 Organization of the thesis

The whole thesis is organized into 5 chapters. Chapter 1 presents an introduction to the theory of clusters and divisible loads. It also gives a brief overview of the related work in the field of RT-DLT. Chapter 2 presents the task model, the system model and the general algorithms used in the scheduling of divisible loads on clusters. Chapter 3 presents the results obtained by using the existing algorithms on a homogeneous cluster where the head node does not take part in the computation of the tasks. This chapter compares the various scheduling algorithms and task partitioning methods to deduce the best one. Chapter 4 shows the derivation of formula for task execution when the head node takes part in computation of tasks. Results are then presented comparing the performance when the head node has and when the head node lacks front-end processing capabilities. Finally, in Chapter 5, the thesis states the conclusion and any modifications or enhancements that can be done in future and next follows the reference section.

Chapter 2

Models

2.1 Introduction

In this section our task and system models are described and the assumptions are stated related to these models. Our task and system models are similar to the task and system models in [9, 10, 11, 13]. We have also described the various algorithms used in scheduling of real-time divisible loads on clusters.

2.2 Task Model

In this paper, we assume aperiodic task model with each task T_i being described by a tuple (A_i, σ_i, D_i), where $A_i \geq 0$ is the task arrival time, $\sigma_i > 0$ is the total data size, and $D_i > 0$ is the relative deadline of the task. Since the tasks have deadlines, the model can be said to assume real-time tasks. The absolute deadline of any task T_i can be given as $A_i + D_i$.

2.3 System Model

We consider a cluster which consists of N processing nodes, denoted by P_1, P_2, ..., P_N, connected to a head node, denoted by P_0, by a switch. We assume a homogeneous model i.e. all processing nodes have the same computational

power. And all the links from the processing nodes to the switch have the same bandwidth. In the first part of our simulation, we assume that the head node doesn't participate in computation while in the later part we assume that it does take part in computation of the tasks. The typical role of the head node is accepting or rejecting the incoming tasks, executing the scheduling algorithm, dividing the workload and distributing the data chunks to the processing nodes. The head node sends every data chunk to be processed by the respective processing node via the switch as different nodes process different data chunks. In case of arbitrarily divisible loads, the tasks and subtasks are independent. There is no task graph relation and hence, when processing such loads, the processing nodes do not communicate with each other.

A cost function $\mathrm{Cp}(\sigma) = \sigma\mathrm{Cps}$ gives the computation time of a load where Cps represents the computation time of a unit workload on a single processing node. A cost function $\mathrm{Cm}(\sigma) = \sigma\mathrm{Cms}$ gives the transmission time of a load where Cms represents the transmission time of a unit workload from head node to a single processing node. Both the cost functions are taken to be linear according to the divisible load theory (DLT) [24].

The following notations partially adopted from [9, 10, 11] are used in the thesis:

- N : Number of processing nodes in the cluster
- P_0 : Head node
- P_1 ,P_2, ..., P_N : the N processing nodes
- Cps : Cost of executing a unit workload on a single processing node
- Cms : Cost of transmitting a unit workload to a single processing node
- $\mathrm{Cp}(\sigma)$: Cost function giving the cost of executing a workload of size σ on a single processing node

- Cm(σ) : Cost function giving the cost of transmitting a workload of size σ on a single processing node

- $\Sigma(\sigma$,n): Execution time function which gives the time taken to complete by a task having data size σ and to which n processing nodes have been assigned.

- (α_1, α_2, ..., α_n) : Data distribution vector where a task of size σ has been assigned n processing nodes and each α_i corresponds to the fraction of the data size σ that is assigned to ith node.

2.4 Algorithms

Any cluster based algorithm is based on three important decision parameters. The first is the scheduling policy which decides the order of execution of the tasks. The second is the task partitioning strategy which decides how to partition the task among a given number of processing nodes. And the third is the node assignment policy which decides how many number of processing nodes should be assigned to each task.

2.4.1 Scheduling Policy

To determine the order of execution of tasks, three scheduling policies are investigated: FIFO, RANDOM and EDF. The FIFO (First In First Out) scheduling policy executes task as per their arrival time. The first task to be executed is the one that has arrived earliest. In case of the RANDOM scheduling policy, the task to be executed next is selected randomly from the available pool of tasks and executed. EDF (Earliest Deadline First) is another well-known scheduling algorithm that executes tasks by their absolute deadlines. The one with the earliest absolute deadline is executed first.

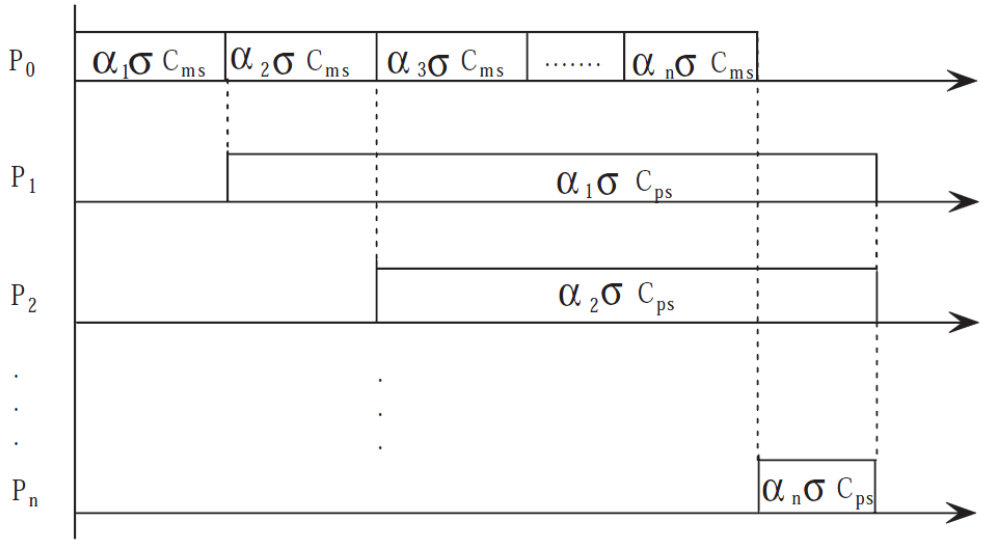

Figure 2.1: Time Diagram for OPR based partitioning

2.4.2 Task Partitioning

Two task partitioning methods are investigated: the OPR (Optimal Partitioning Rule) and the EPR (Equal Partitioning Rule). OPR is based on DLT and hence all the nodes allocated to a particular task complete execution at the same time. EPR divides a task into a number of subtasks of equal size and they are assigned to different nodes for execution. Here we present in brief the time diagrams and execution time function for both OPR and EPR partitioning established in [9, 10, 11]:

OPR Partitioning

In the time diagram of figure 2.1, we notice that all the subtasks in the different processing nodes complete their execution at the same time. The execution time function for OPR partitioning was derived to be:

$\Sigma(\sigma,n) = (1-\beta)/(1-\beta^n) * \sigma * (Cms+Cps)$ .....(1)

where $\beta = Cps/(Cms+Cps)$ .....(2)

EPR Partitioning

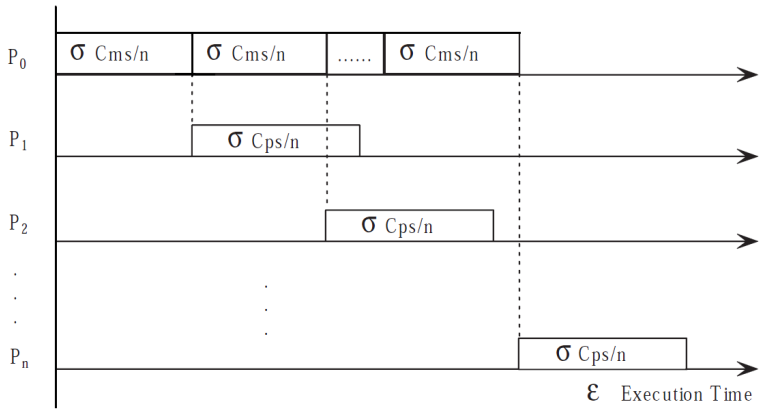

Figure 2.2: Time Diagram for EPR based partitioning

In the time diagram of figure 2.2, we notice that all of the data chunks are equal in size. The execution time function was derived to be:

$\Sigma(\sigma,n) = \sigma Cms + (\sigma Cps/n)$ .....(3)

2.4.3 Node Assignment

There are two ways of node assignment. Either we can assign all N nodes to a task and finish the task as early as possible or we may assign the minimum number of nodes required to finish the task in time and save resources for newer tasks. In our simulation, we use the former method of node assignment by assigning all N nodes to a task.

Based on these three decision parameters, we get various algorithms like FIFO-EPR-AN, FIFO-EPR-MN, FIFO-OPR-AN, FIFO-OPR-MN, RANDOM-EPR-AN, RANDOM-EPR-MN, RANDOM-OPR-AN, RANDOM-OPR-MN, EDF-EPR-AN, EDF-EPR-MN, EDF-OPR-AN and EDF-OPR-MN. In our work, we

have assumed, in all cases that the task is assigned all the N nodes. In future, our inclusion of head node having front end processing capability could be studied for the other node assignment policy (MN) where each task is assigned the minimum number of nodes (n^{min}) it needs to finish by its deadline.

2.5 Conclusion

The problem of scheduling real-time divisible loads on clusters is basically approached through 3 critical steps - choice of scheduling algorithm, choice of task partitioning strategy and choice of node-assignment policy. This gives rise to various combinations of algorithms. We have, in this thesis, examined 6 algorithms resulting from variations of the first 2 steps stated above - EDF-OPR-AN, EDF-EPR-AN, FIFO-OPR-AN, FIFO-EPR-AN, RANDOM-OPR-AN, RANDOM-EPR-AN.

Chapter 3

Simulation Setup and Results

3.1 Introduction

In this chapter, we have first described our simulation setup - the various assumptions made, the parameters taken and their distribution models used. First we have implemented the 3 scheduling algorithms EDF, FIFO and Random using both EPR and OPR partitioning methods. We assumed that all N nodes are assigned to a task. Hence, in this chapter we have evaluated and compared the performance of these 6 algorithms EDF-OPR-AN, EDF-EPR-AN, FIFO-OPR-AN, FIFO-EPR-AN, RANDOM-OPR-AN, RANDOM-EPR-AN on a homogeneous cluster where the head node lacks front-end processing capability. We have taken the evaluation parameter as the Task-Rejection-Ratio. The Task-Rejection-Ratio is the ratio of the number of tasks rejected to the total number of tasks evaluated. First we have shown the result of our experiment which gives us the number of processing nodes our cluster should have, given the other parameters we have taken. Then we have shown the results comparing the performance of the above 6 algorithms. In the next chapter, we shall see the performance of the scheduling when the head-node takes part in the actual computation of the tasks.

3.2 Simulation Setup

We have taken our task model as (A, σ, D) and system model as (N, Cms, Cps) as described in the section 2.2 and 2.3. Our simulation parameters have been partially derived from Lin et al. [9, 10, 11]. We have basically assumed the values of 5 independent parameters avgσ, DCRatio, Cms, Cps, lambdainv.

- avgσ is the average load size of the tasks
- DCRatio is the ratio of avgD to Σ(avgσ,N). avgD is the average relative deadline of the tasks and $\Sigma(\sigma$,n) is the execution time function as given in equation (1) in section 2.4.2.
- Cms and Cps are the same terms as defined in section 2.3.
- lamdainv is the mean of the inter-arrival time of the tasks. Taking a cue from [9, 10, 11], we have assumed avgσ to be 100, DCRatio to be 2, Cms as 1, Cps as 100 and lambdainv as 400.
- The inter-arrival time in our simulation follows an exponential random distribution with mean as lamdainv.
- The load size of the tasks σ_i are normally distributed with both mean and standard deviation equal to avgσ.
- The task relative deadlines are assumed to be uniformly distributed in the range [avgD/2, 3avgD/2]. avgD is calculated from DCRatio and Σ(avgσ,N) as values of both these terms are known.

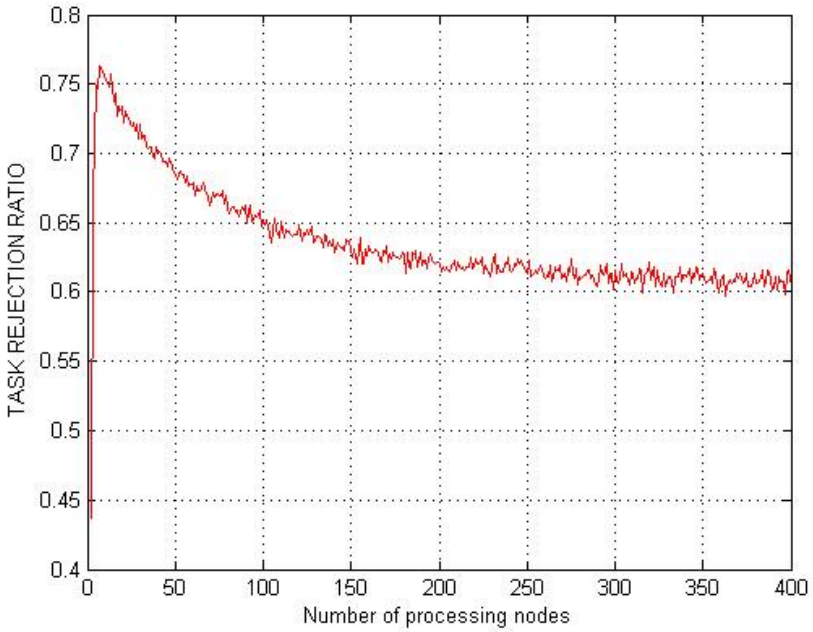

Figure 3.1: Analysis of effect of number of nodes over TRR for EDF-OPR-AN

3.3 Simulation Results

3.3.1 Effect of number of nodes on Task-Rejection-Ratio (TRR)

From figure 3.1, we observe that beyond a certain point (in this case at N = 100), the number of processing nodes that the cluster has doesn't appreciably affect the Task-Rejection-Ratio. We find that beyond N=100 (for the parameters of the simulation that we have taken), the graph of Task-Rejection-Ratio is confined within TRR=0.6. Hence, we observe that taking too many nodes doesn't necessarily improve the performance and we know that more the number of nodes, more will be the cost of hardware realization of the cluster. Hence, in all of our further simulations, we have taken N to be 100.

3.3.2 Comparision of Algorithms

Comparision of EDF-OPR-AN, FIFO-OPR-AN, RANDOM-OPR-AN

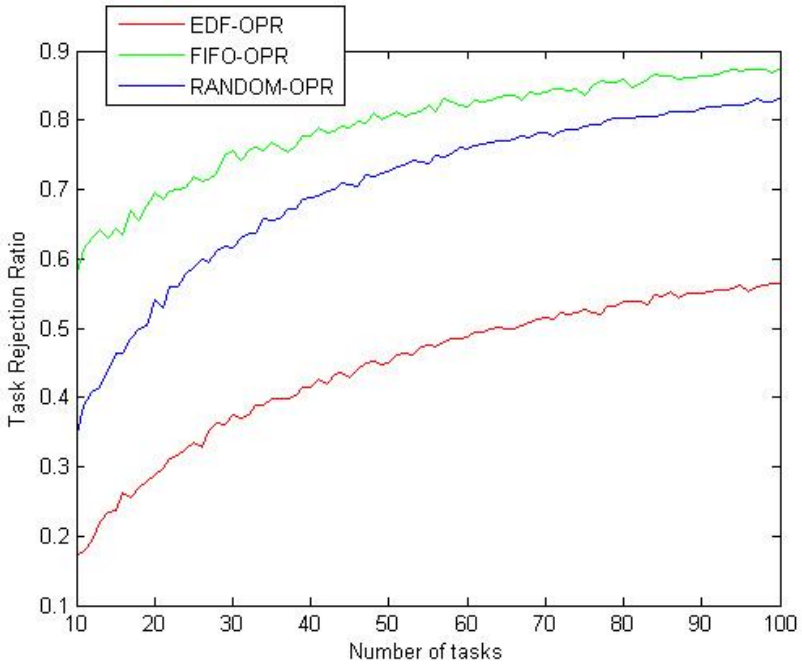

Figure 3.2: Analysis of the 3 scheduling algorithms EDF, FIFO and RANDOM for OPR partitioning on a homogeneous cluster with head-node lacking front-end processing capability

From Figure 3.2, we observe that the graph of EDF-OPR-AN lies below the graphs of FIFO-OPR-AN and RANDOM-OPR-AN. Hence, we conclude that EDF is better than the other two algorithms for OPR partitioning as it has less Task-Rejection-Ratio throughout. Similarly, it can be said that RANDOM scheduling is better than FIFO for OPR partitioning.

Comparision of EDF-EPR-AN, FIFO-EPR-AN, RANDOM-EPR-AN

From Figure 3.3, we observe that the graph of EDF-EPR-AN lies below the graphs of FIFO-EPR-AN and RANDOM-EPR-AN. Hence, we conclude that EDF is better than the other two algorithms for EPR partitioning as it has

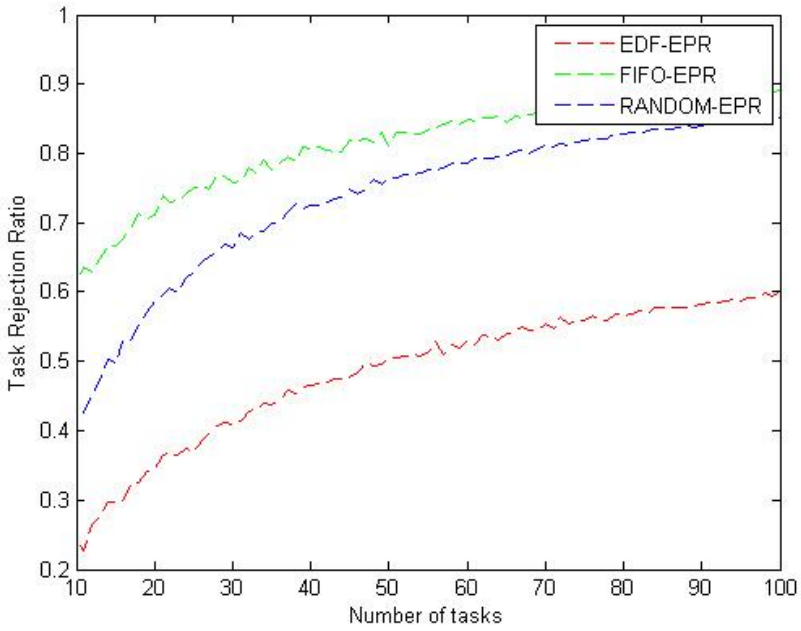

Figure 3.3: Analysis of the 3 scheduling algorithms EDF, FIFO and RANDOM for EPR partitioning on a homogeneous cluster with head-node lacking front-end processing capability

less Task-Rejection-Ratio throughout. Similarly, it can be said that RANDOM scheduling is better than FIFO for EPR partitioning.

Comparision of EDF-OPR-AN and EDF-EPR-AN

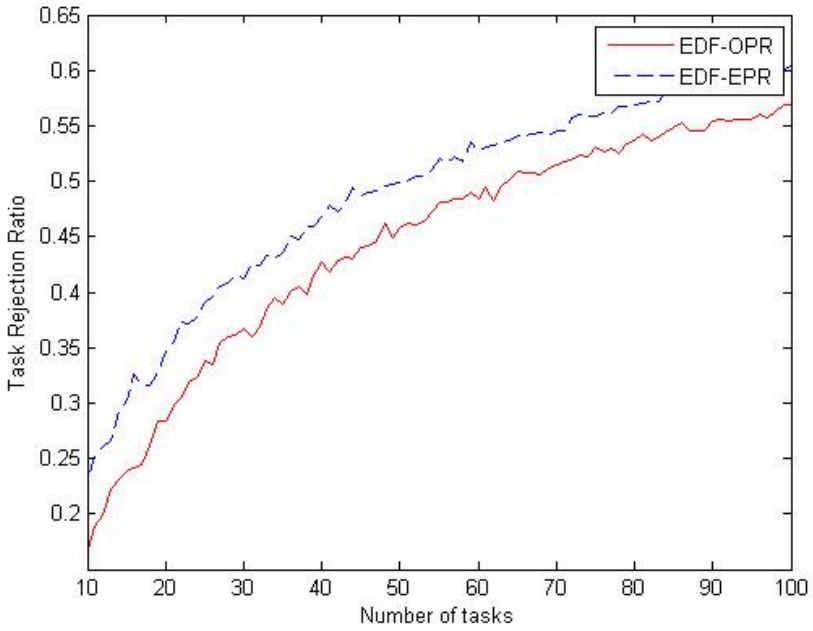

Figure 3.4: Analysis of the OPR and EPR partitioning for EDF scheduling on a homogeneous cluster with head-node lacking front-end processing capability

From Figure 3.4, we observe that the graph of EDF-OPR-AN lies below the graph of EDF-EPR-AN. Hence, we conclude that the OPR partitioning gives better performance than EPR partitioning as it has less Task-Rejection-Ratio throughout. This supports the existing proofs that DLT is advantageous in scheduling of real-time loads on clusters.

Comparision of all the six algorithms

From Figure 3.5, we observe that the graph of EDF scheduling is better than Random scheduling which, in turn, is better than FIFO scheduling for both OPR and EPR partitioning of tasks. We also find that the OPR partition-

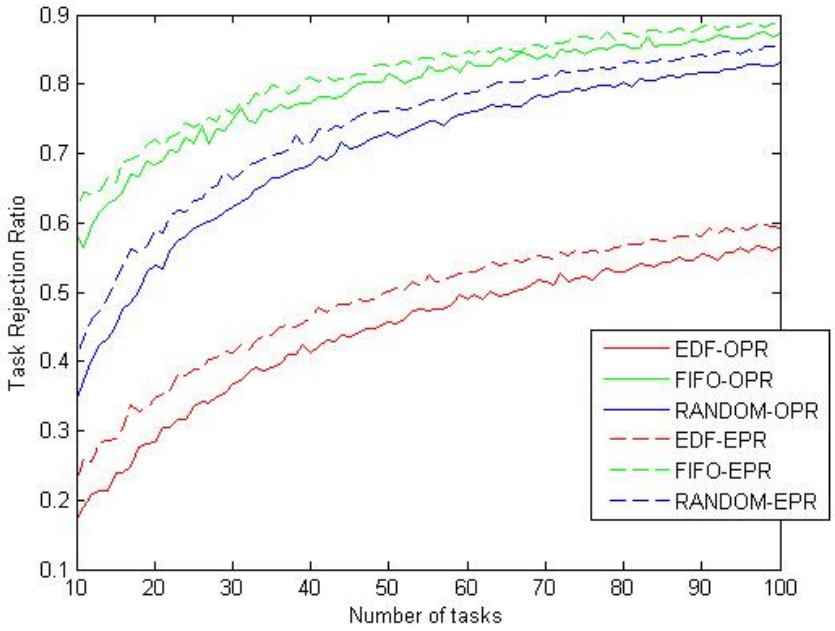

Figure 3.5: Analysis of the performance of all the 6 algorithms on a homogeneous cluster with head-node lacking front-end processing capability

ing gives better performance than EPR partitioning for all the 3 algorithms EDF, FIFO, RANDOM. Hence, we conclude that EDF-OPR-AN is the best algorithm among these 6 for the scheduling of real-time divisible loads on a homogeneous cluster where the head-node lacks front-end processing capability.

3.4 Conclusion

From the findings of the above simulations, we conclude that the number of processing nodes in a cluster affects the scheduling performance only up to a certain point beyond which, it remains more or less the same. We also conclude that EDF always outperforms FIFO and RANDOM scheduling algorithms. Similarly, OPR partitioning is better than EPR partitioning which supports the already established advantageous nature of DLT in the context of scheduling real-time workloads on clusters.

Chapter 4

Head Node having front-end processing capabilities

4.1 Introduction

In the previous chapter, we found that the EDF scheduling is better than FIFO and RANDOM scheduling and that OPR partitioning always outperforms EPR partitioning. Hence, here we study and compare the performance of only EDF-OPR-AN for the 2 scenarios - one where the head node in a homogeneous cluster does not take part in the actual computation of the tasks and the other where the head node in a homogeneous cluster participates in the actual computation of the tasks. In this chapter, we have first derived the new execution time formula for the new scenario from its time diagram. Then we have described the simulation setup and finally, we have shown the simulation results.

4.2 Derivation of new execution time function

In the time diagram of figure 4.1, P_0 is the head node while P_1, P_2, ..., P_n are the other n processing nodes of the cluster. The other symbols α_i, σ, Cms,

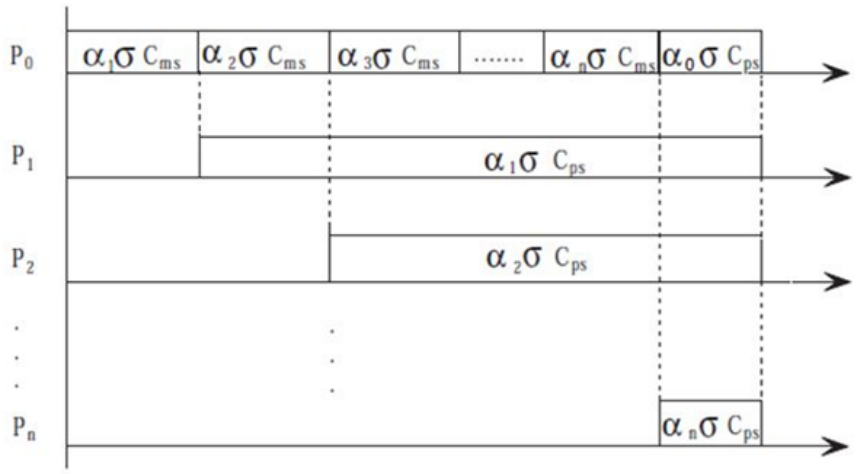

Figure 4.1: Time-Diagram for OPR partitioning when the head node (P_0) takes part in the actual computation of the task

Cps are the same as described in section 2.3.

We find that the time-diagram of figure 4.1 differs from the time-diagram of figure 2.1 in the respect that, in figure 4.1, after the transmission of the data chunk of the n^{th} processing node, the head node too executes a data chunk of size $\alpha_0\sigma$. This changes the entire data distribution vector and gives rise to a new formula for the execution time function which is derived as follows:

From the time diagram of figure 4.1, we find the following:

$\Sigma(\sigma,n)=\alpha_1\sigma Cms + \alpha_1\sigma Cps$ .....(4)

$\Sigma(\sigma,n)=(\alpha_1+\alpha_2)\sigma Cms + \alpha_2\sigma Cps$ .....(5)

...

$\Sigma(\sigma,n)=(\alpha_1+\alpha_2+.....+\alpha_n)\sigma Cms + \alpha_n\sigma Cps$ .....(6)

$\Sigma(\sigma,n)=(\alpha_1+\alpha_2+.....+\alpha_n)\sigma Cms + \alpha_0\sigma Cps$ .....(7)

Hence, from above equations (6) and (7), we get $\alpha_0 = \alpha_n$ .....(8)

From equations (4) and (5), we get $\alpha_2=\beta\alpha_1$, where β=Cps/(Cps+Cms) .....(9)

Similarly, we will get $\alpha_3=\beta\alpha_2$, ..., $\alpha_n=\beta\alpha_{n-1}$ .....(10)

Now as α_0, α_1, ..., α_n are fractions of the total data size, their summation will result in 1. Hence, we have

$\alpha_0 + \alpha_1 + \beta\alpha_1 + \beta^2\alpha_1 + ... + \beta^{n-1}\alpha_1 = 1$ .....(11)

From equations (8), (10) and (11), we get $\alpha_1 = 1/(\beta^{n-1} + (1-\beta^n)/(1-\beta))$ .....(12)

Hence, from equations (4) and (12), we get

$\Sigma(\sigma, n) = 1/(\beta^{n-1} + (1-\beta^n)/(1-\beta)) * \sigma * (Cms + Cps)$ .....(13)

This is the new formula for the execution time function in contrast to that of equation (1).

4.3 Simulation Setup

The simulation setup for this scenario, where the head node has front-end processing capability is the same as that described in section 3.2 with one major difference -

- $\Sigma(avg\sigma, N)$ is calculated here using the formula of equation (13) derived above and not the formula of equation (1). Hence the value of $\Sigma(avg\sigma, N)$ is different here which leads to a different value of avgD which ultimately leads to a new distribution for the task relative deadlines.

4.4 Conclusion

In figure 4.2, we observe that the graph of EDF-OPR-AN for the case when the head node participates in the actual computation of the tasks lies below the graph of EDF-OPR-AN for the case when the head node doesn't participate in the actual computation of the tasks. We observe the same thing in figure 4.3 where the parameter is the Total Execution Time instead of the Task-Rejection-Ratio. Hence we can conclude that the scheduling of divisible loads shows better performance when the head node takes part in the actual computation of the tasks (utilizing its idle time after the transmission of the n^{th} data chunk to the n^{th} processor) as it gives lesser Task-Rejection-Ratio and also lesser Total Execution Time as compared to the scenario when the head node of the cluster doesn't participate in the actual computation of the tasks.

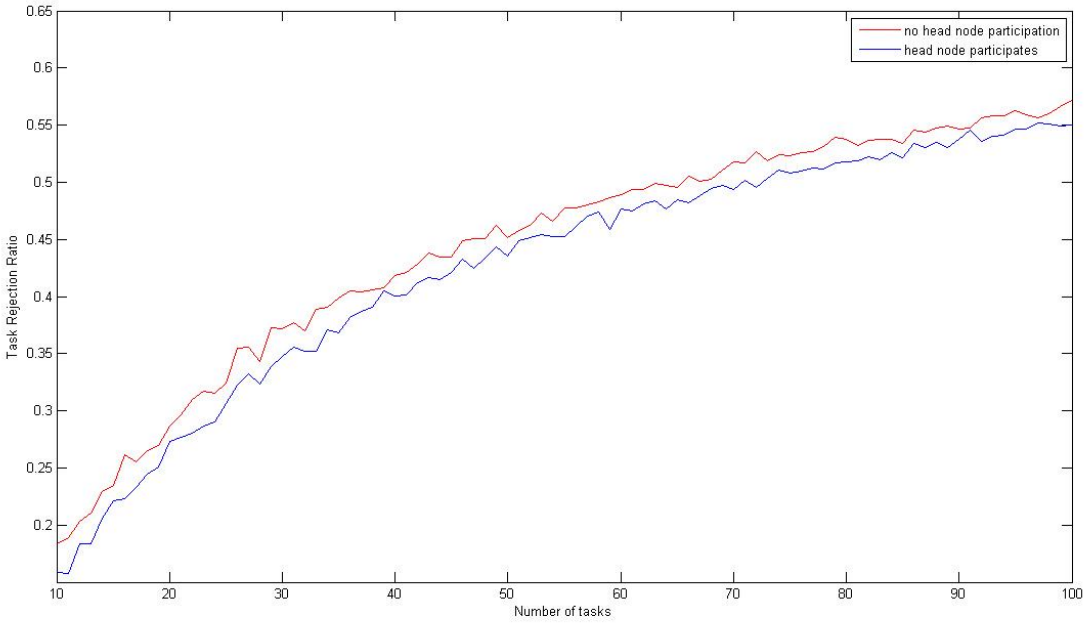

Figure 4.2: Analysis of EDF-OPR-AN for both the cases when head node does and doesn't participate in the actual computation of the tasks with Task-Rejection-Ratio as the parameter

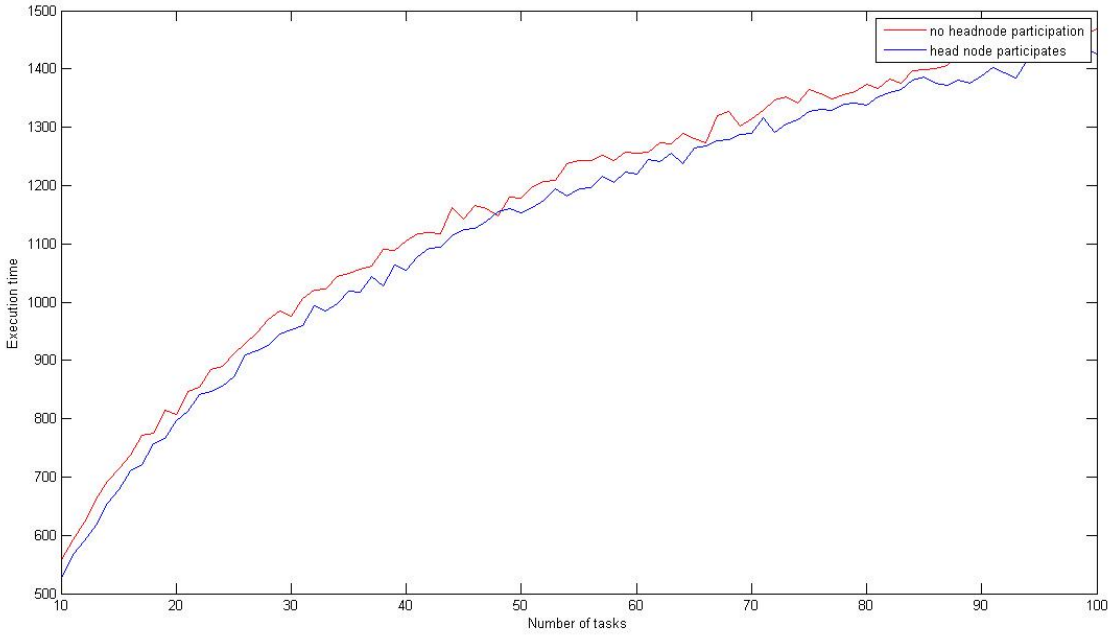

Figure 4.3: Analysis of EDF-OPR-AN for both the cases when head node does and doesn't participate in the actual computation of the tasks with Task-Rejection-Ratio as the parameter

Chapter 5

Conclusions and Future Work

5.1 Conclusions

The problem of scheduling of real-time divisible loads on clusters proceeds basically through 3 steps or 3 design decisions - the choice of scheduling algorithm, the choice of task partitioning method and the choice of node assignment policy. In this thesis, we have examined the performance of the algorithms that result from the variation of the first two design decisions on two different kinds of homogeneous clusters - one where the head node lacks front-end processing capability and one where the head node does have front-end processing capability. As far as the node assignment policy is considered, we have taken only the case where all N nodes of the cluster are assigned to a task.

From our results, we conclude that irrespective of the task partitioning method chosen, the Earliest-Deadline-First(EDF) algorithm always performs better than the Random and First-In-First-Out(FIFO) scheduling algorithms. Also it was found that the OPR partitioning of the tasks always gives better performance than EPR partitioning irrespective of the scheduling algorithm chosen. This supports the established notion that application of Divisible Load Theory is advantageous in the context of scheduling of real-time workloads in cluster computing environments. We also concluded from our findings that

increase in the number of processing nodes in the cluster betters the scheduling performance only up to a certain point beyond which, the performance remains more or less the same. Hence, it is beneficial to derive the ideal node size depending upon the constraints of the scenario otherwise, an arbitrarily high number of processing nodes may compound hardware cost and an arbitrarily low number of processing nodes may not lead to optimum performance. Finally we concluded that even though not large, the participation of the head node in the actual computation of the tasks (utilizing its idle time after the transmission of the n^{th} data chunk to the n^{th} processing node) gives better scheduling performance as compared to the case where the head node doesn't take part in the actual computation of the tasks.

5.2 Future Work

In this thesis, we have taken into consideration only the variations of the first two design decisions - choice of scheduling algorithm and choice of task partitioning strategy. It can be extended to include variation in the choice of node assignment policy - whether to assign all N nodes or only the minimum number of nodes n^{min} that is required by the task to finish by its deadline. Also, in case of scheduling algorithms, we have studied only EDF, FIFO and RANDOM. It can be extended to include another scheduling algorithm - Maximum Workload-derivative First(MWF). Hence, the following additional algorithms could be examined to extensively study the effect of head node taking part in actual computation of the tasks - EDF-OPR-MN, EDF-EPR-MN, FIFO-OPR-MN, FIFO-EPR-MN, RANDOM-OPR-MN, RANDOM-EPR-MN, MWF-OPR-AN, MWF-OPR-MN, MWF-EPR-AN, MWF-EPR-MN. Also, in this thesis, we examined the effect of head node taking part in task computation only for homogeneous cluster. This can be examined further for heterogeneous clusters too.

Bibliography

[1] X. Lin, J. Deogun, Y. Lu and S. Goddard. Multi-round Real-Time Divisible Load Scheduling for Clusters. In *Proceedings of 15th International Conference on High Performance Computing (HiPC)*, pp. 196-207, Bangalore, India, December 2008.

[2] A. Mamat, Y. Lu, J. Deogun and S. Goddard. Real-Time Divisible Load Scheduling with Advance Reservation. In *Proceedings of the Euromicro Conference on RealTime Systems (ECRTS)*, pp. 37-46, Prague, Czech Republic, July 2008.

[3] S. Chuprat and S. Baruah. Deadline-based scheduling of divisible real-time loads. In *Proceedings of the ICSA 20th International Conference on Parallel and Distributed Computing Systems (PDCS)*, pp. 7-12, Las Vegas, Nevada, September, 2007.

[4] G.Manimaran and C. S. R. Murthy. An efficient dynamic scheduling algorithm for multiprocessor realtime systems. *IEEE Transaction on Parallel and Distributed Systems*, 9(3):312-319, 1998.

[5] W. Lee, S. J. Hong, and J. Kim. On-line scheduling of scalable real-time tasks on multiprocessor systems. *Journal of Parallel and Distributed Computing*, 63(12):1315–1324, 2003.

[6] S. Collette, L. Cucu, and J. Goossens. Algorithm and complexity for the global scheduling of sporadic tasks on multiprocessors with work-limited

parallelism. In *Proceedings of the 15th International Conference on Real-Time Systems (RTNS)*, pp. 123-128, Nancy, France, March 2007.

[7] S. Collette, L.Cucu, and J.Goossens. Integrating Job Parallelism in Real-Time Scheduling Theory. *Information Processing Letters*, 106(5):180-187, May, 2008.

[8] S. Chuprat, S. Salleh and S. Goddard. Real-Time Divisible Load Theory: A Perspective. In *Proceedings of International Conference on Parallel Processing Workshops*, pp. 6-10, 2009.

[9] X. Lin, Y. Lu, J. Deogun, and S. Goddard. Real-time divisible load scheduling for cluster computing. *Technical Report UNL-CSE-2006-0016*, Department of Computer Science and Engineering, The University of Nebraska at Lincoln, 2006.

[10] X. Lin, Y. Lu, J. Deogun, and S. Goddard. Real-time divisible load scheduling for clusters. In *Proceedings of the Real-Time Systems Symposium Work-In-Progress Session*, pages 912, Rio de Janerio, December 2006.

[11] X. Lin, Y. Lu, J. Deogun, and S. Goddard. Real-time divisible load scheduling for cluster computing. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS)*, pp. 303-314, Bellevue, Washington, April, 2007.

[12] X. Lin, Y. Lu, J. Deogun, and S. Goddard. Real-time divisible load scheduling with different processor available times. In *Proceedings of The International Conference on Parallel Processing (ICPP)*, Xian, China, September 2007.

[13] X. Lin, A. Mamat, Y. Lu, J. Deogun, and S. Goddard. Realtime scheduling of divisible loads in cluster computing environments. *Journal of Parallel and Distributed Computing*, 70(3):296-308, 2010.

[14] A. Mamat, Y. Lu, J. Deogun, and S. Goddard. An efficient algorithm for real-time divisible load scheduling. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, Stockholm, Sweden, April, 2010.

[15] Compact Muon Solenoid (CMS) Experiment for the Large Hadron Collider at CERN (European Lab for Particle-Physics), *Cms web page, http://cmsinfo.cern.ch/Welcome.html/.*

[16] ATLAS (AToroidal LHC Apparatus) Experiment, CERN (European Lab for Particle Physics), *Atlas web page, http://atlas.ch/.*

[17] S. Chuprat. Divisible load scheduling of real-time task on heterogeneous clusters. In *Proceedings of the International Symposium in Information Technology (ITSim)*, 2:721-726, 15-17 June, 2010.

[18] S. Chuprat, S. Salleh and S. Baruah. Evaluation of a linear programming approach towards scheduling divisible real-time loads. In *Proceedings of the International Symposium on Information Technology (ITSim, co-sponsored by the IEEE)*, 1:455-462, Kuala Lumpur, Malaysia. August 2008.

[19] S. Chuprat and S. Baruah. Scheduling Divisible RealTime Loads on Clusters with Varying Processor Start Times. In *Proceedings of the IEEE 14th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pp. 15-24, Kaohsiung, Taiwan. August 2008.

[20] S. Chuprat and S. Baruah. Real-Time Divisible Load Theory: Incorporating Computation Costs. In *Proceedings of the IEEE 17th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 1:28-31, 33-37. August 2011.

[21] V. Bharadwaj, B. Veeravalli, and W. H. Min. Scheduling divisible loads on heterogeneous linear daisy chain networks with arbitrary processor release times. *IEEE Trans. on Parallel and Dist. Systs.*, 15(3):273-288, 2004.

[22] M. Drozdowski, and P. Wolniewicz. Optimum Divisible Load Scheduling on Heterogeneous Stars with Limited Memory. *European Journal of Operational Research*, 172(2):545-559, July 2006.

[23] H. Gonzalez-Velez and M. Cole. Adaptive Statistical Scheduling of Divisible Workloads in Heterogeneous Systems. *Journal of Scheduling*, 2009.

[24] B. Veeravalli, D. Ghose, and T. G. Robertazzi. Divisible load theory: A new paradigm for load scheduling in distributed systems. *Cluster Computing*, 6(1):717, 2003.

[25] J. Sohn, T. G. Robertazzi, and S. Luryi. Optimizing computing costs using divisible load analysis. *IEEE Transactions on Parallel and Distributed Systems*, 9(3):717, 1998.