Survey of Real Time Divisible Load Scheduling Algorithms in computational grid

Mohamed Youssri A. El Nahas [1], Nahed M. El Desouky [2], Sahar A. Gomaa[2]and Naglaa Mostafa [2]
[1]Faculty of Engineering , Al-Azhar University(girls) Cairo, Egypt.
[2]Departement of Mathematics, Computer Branch, Faculty of Science ,
Al-Azhar University(girls), Cairo, Egypt

Abstract - *Cluster computing has emerged as a new paradigm for solving large-scale problems. These workloads represent a broad variety of real-world applications in cluster and grid computing, such as BLAST (Basic Local Alignment Search Tool) [2], a bioinformatics application, and high energy and particle physics applications in ATLAS (A Toroidal LHC Apparatus) [6] and CMS (Compact Muon Solenoid) [10] projects. To provide performance guarantees in cluster computing environments, various real-time scheduling algorithms and workload models have been investigated. Computational loads that can be arbitrarily divided into independent pieces represent many real-world applications. Divisible load theory (DLT) provides insight into distribution strategies for such computations. However, the problem of providing performance guarantees to divisible load applications has not yet been systematically studied. This paper provides a survey and compares different algorithms for a cluster environment that provide a solution for the real time divisible load applications. And provide different parameters that affect the performance of these algorithms and scenarios when the choice of these parameters has significant effects are studied. It provides a taxonomy of the different scheduling methods, and considers the various performance metrics that can be used for comparison purposes*.

Keywords: grid computing, divisible load theory.

1 Introduction

Real-time Divisible Load Theory (RT-DLT) holds great promise for modeling an emergent class of massively parallel real-time workloads. However, the theory needs strong formal foundations before it can be widely used for the design and analysis of hard real-time safety-critical applications. In[1] the general problem of obtaining such formal foundations, by generalizing and extending recent results and concepts from multiprocessor real-time scheduling theory.

Cluster computing has become an important paradigm [2] for solving large-scale problems. However, as the size of a cluster increases, so does the complexity of resource management and maintenance. Therefore, automated performance control and resource management are expected to play critical roles in sustaining the evolution of cluster computing.

Real-time divisible load scheduling is a well researched area [3, 4, 5, 6, 7, 8]. Focusing on satisfying QoS (Quality of services), providing real-time guarantees, and better utilizing cluster resources, existing approaches give little emphasis to scheduling efficiency. They assume that scheduling takes much less time than the execution of a task, and thus the scheduling overhead is ignored. When the processors in a cluster platform all become available at the same instant in time, the issue of scheduling a real-time divisible workload on such platforms is pretty well understood. However, the reality in many multiprocessor environments is that all the processors do not become available to a given workload at the same instant (perhaps because some of the processors are also being used for other purposes).

Broadly speaking, computational loads submitted to a cluster are structured in two primary ways: *indivisible* and *divisible*. An indivisible load is essentially a sequential job and thus must be assigned to a single processor. The divisible loads are comprised of tasks that can be executed in parallel and can be further divided into two categories: *modularly divisible* and *arbitrarily divisible* loads. *Modularly divisible* loads are divided a priori into a certain number of subtasks and are often described by a task (or processing) graph. *Arbitrarily divisible* loads can be partitioned into an arbitrarily large number of load fractions. Examples of arbitrarily divisible loads can be easily found in high-energy and particle physics as well as bio-metrics. For example, the CMS (Compact Muon Solenoid) [18] and ATLAS (A Toroidal LHC Apparatus) [19] projects, which are asso-ciated with the LHC (Large Hadron Collider) at CERN (European Laboratory for Particle Physics), execute cluster-based applica-tions with arbitrarily divisible loads.

There are three important decisions for an algorithm of real-time scheduling to schedule divisible loads. The first is to adopt a scheduling policy to determine the order of execution for tasks. The second decision is to determine the number of processing nodes (n) to allocate to each task and the third is to choose a strategy to partition the task among the allocated n nodes.

Description of the distributed system and assumption used by different algorithms of real time divisible load are discussed in section 2. Section 3 studies different scheduling policies that classify the algorithms used in real time divisible load. Different algorithms that are used to decide the number of nodes must be assigned to each task and method of portioning the task itself is discussed in section 4. The metrics used to measure the real time performance of different scheduling algorithms are explained in section 5 .Conclusion is given in section 6**.**

1.1 Instructions for authors

An electronic copy of your *full camera-ready paper* must be uploaded (in PDF format) to Publication Web site before the announced deadline. Please follow the submission instructions shown on the web site. The URL to the website is included in the notification of acceptance that has been emailed to you by Prof. Arabnia.

2 Tasks and System Assumption

The model of the investigated system used by different algorithms and the assumptions are as follow:

System model: a cluster consists of a head node, denoted by p_0, and connected via a switch to *n* processing nodes, denoted by $p_1, p_2, p_3, ...p_n$. And assuming that all processing nodes have the same computational power and all links from the switch to the processing nodes have the same bandwidth. The system model assumes a typical cluster environment in which the head node does not participate in computation. The role of the head node is to accept or reject incoming tasks, execute the scheduling algorithm, divide the workload and distribute data chunks to processing nodes. Since different nodes process different data chunks, the head node sequentially sends every data chunk to its corresponding processing node via the switch. And data transmission does not happen in parallel, although it is straightforward to generalize this model and include the case where some pipelining of communication may occur. There is an assumption for the divisible loads that is task and subtasks are independent. Therefore, there is no need for processing nodes to communicate with each other.

According to divisible load theory, linear models are used to represent processing and transmission times [10]. In the simplest scenario, the computation time of a load σ is calculated by a cost function $cp\ (\sigma) = \sigma c_{ps}$, where c_{ps} represents the time to compute a unit of workload on a single processing node. The transmission time of a load σ is calculated by a cost function $cm(\sigma) = \sigma c_{ms}$, where c_{ms} is the time to transmit a unit of workload from the head node to a processing node.

Tasks Assumption: Assuming that a real time aperiodic task model in which each a periodic task *Ti* consists of a single invocation specified by the tuple (A_i, σ_i, D_i), where $A_i \geq 0$ is the arrival time of the task, $\sigma_i > 0$ is the total data size of the task, and $D_i > 0$ is its relative deadline. The absolute deadline of the task is given by $A_i + D_i$ **.** Task execution time is dynamically computed based on total data size σ_i , resources allocated (i.e., processing nodes and bandwidth) and the partitioning method applied to parallelize the computation.

3 Scheduling Policies

There are three scheduling policies to determine the execution order of tasks that are investigated in different algorithms: FIFO, EDF (earliest deadline first) and MWF (Maximum Workload derivative First)[1,2,9,10,12,20]. The FIFO scheduling algorithm executes tasks following their order of arrival. EDF, a well-known real-time scheduling algorithm, orders tasks by their absolute deadlines. MWF is a real-time scheduling algorithm for divisible tasks. The main rules of MWF are:

1) A task with the highest workload derivative (DC_i) is scheduled first.

2) The number of nodes allocated to a task is kept as small as possible ($n_i^{\min}$) without violating its deadline. Here, we review how MWF determines task execution order and define the workload derivative metric, DC_i, where $W_i(n)$ is used to represent the workload (cost) of a task T_i when n processing nodes are assigned to it.

$$DC_i = W_i\left(n_i^{\min}+1\right) - W_i\left(n_i^{\min}\right) \qquad (1)$$

That is, $W_i(n) = n \times \xi(\sigma_i, n)$, where $\xi(\sigma_i, n)$ denotes the task's execution time. Therefore, DC_i is the derivative of the task workload $W_i(n)$ at $n_i^{\min}$ (the minimum number of nodes needed by *Ti* to meet its deadline).

4 Real time divisible algorithms

Different algorithms that investigate the real time divisible load are studied during the past few years. OPR which searches optimally to find the minimum number of processing nodes that satisfies the real time requirement of the load that is proposed in [1, 2, 9, 10, 20]. Another way to solve this problem follows the idea of equally portioning the load among the processing nodes that presented in [1, 2, 9, 10]. Task Waiting Queue (TWQ) algorithms are divisible load

scheduling algorithms [3, 4, 5, 6, 7], that perform the schedule ability test. The admission controller generates a new schedule for the newly arrived task and all tasks waiting in TWQ, this decision module is referred to as the admission controller. When processing nodes become available, the dispatcher module partitions each task and dispatches subtasks to execute on processing nodes. And finally there are algorithms depend on the time which processing nodes are available at it. There are two algorithms of this idea first when processing nodes have equal ready and when processing nodes have different ready time, these algorithms presented in [7,14, 15]. In the following subsections a brief details of these algorithms are presented.

4.1 Optimal Partitioning Rule (OPR) Algorithm

Divisible load theory states that optimal execution time is obtained for a divisible load if all processing nodes allocated to the task complete their computation at the same time instant [11]. This is called the *Optimal Partitioning Rule* (OPR). In divisible load theory, normally all *n* nodes of a cluster are allocated to a task. Then, following the OPR, the task load is partitioned such that all nodes finish processing at the same time. In contrast to this approach, first computing the minimum number of processing nodes needed to meet the task's deadline given its schedule, and then partition the task following the OPR (using at least the minimum number of nodes required to meet the deadline). The execution time of a task is then trivially computed as the difference between its completion and start times. The following notations, partially adopted from [11], are used in these computations.

- $T=(A,\sigma,D)$: A divisible task, where *A* is the arrival time, σ is the data size, and *D* is the relative deadline.
- $\alpha=(\alpha_1,\alpha_2,.....,\alpha_n)$: Data distribution vector, where *n* is the number of processing nodes allocated to the task, α_j is the data fraction allocated to the j^{th} node, i.e., $\alpha_j\sigma$, is the amount of data that is to be transmitted to the j^{th} node for processing, $0\prec\alpha_j\leq 1$ and $\sum_{j=1}^{n}\alpha_j=1$.
- c_{ms} : Cost of transmitting a unit workload.
- c_{ps} : Cost of processing a unit workload.

The following cost functions describe that: the data transmission time on the j^{th} link is $c_m(\alpha_j\sigma)=\alpha_j\sigma c_{ms}$ and the data processing time on the j^{th} node is $c_p(\alpha_j\sigma)=\alpha_j\sigma c_{ps}$.

Figure 1 shows an example task execution time diagram following OPR when *n* nodes are allocated to process the task. Let ξ denote *Task Execution Time*, which is a function of σ and *n*.

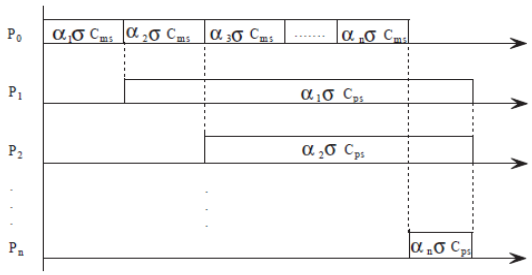


Figure 1: Time Diagram for OPR-Based Partitioning**.**

By analyzing the diagram, we have

$$\xi(\sigma,n)=\alpha_1\sigma c_{ms}+\alpha_1\sigma c_{ps} \quad (2)$$

$$=(\alpha_1+\alpha_2)\sigma c_{ms}+\alpha_2\sigma c_{ps} \quad (3)$$

$$=(\alpha_1+\alpha_2+\alpha_3)\sigma c_{ms}+\alpha_3\sigma c_{ps} \quad (4)$$

$$\cdots$$

$$=(\alpha_1+\alpha_2+\alpha_3+.....+\alpha_n)\sigma c_{ms}+\alpha_n\sigma c_{ps}\ . \quad (5)$$

To specify the minimum number $n^{\min}$ of nodes that required to meet a task's deadline**,** **a**ssuming that the task $T(A,\sigma,D)$ has a start time s, then the task completion time is C (n) = s + $\xi(\sigma,n)$, which must satisfy the constraint that C(n) ≤ A + D. Lin et al. [14, 15, 16] derived the task execution time function $\xi(\sigma,n)$ and the minimum number $n^{\min}$ of nodes that the task needs at time *s* to meet its deadline are

$$\xi(\sigma,n)=\frac{1-\beta}{1-\beta^n}\sigma(c_{ms}+c_{ps}) \quad (6)$$

$$n^{\min}=\left\lceil\frac{\ln\gamma}{\ln\beta}\right\rceil \quad (7)$$

where $\beta=\frac{c_{ps}}{c_{ms}+c_{ps}}$ and $\gamma=1-\frac{\sigma c_{ms}}{A+D-s}$.

4.2 Equal Partitioning Rule (EPR) Algorithm

Equal Partitioning Rule (EPR) is based on a common practice of dividing a task into *n* equal-sized subtasks when the task is to be processed by *n* nodes. An example task execution time diagram following the EPR is shown in figure 2. By analyzing the diagram, we have

$$\xi(\sigma,n)=\sigma c_{ms}+\frac{\sigma c_{ps}}{n}\ . \quad (8)$$

Similar to the analysis for DLT-based OPR, Lin et al. [1, 2 ,9, 10] derived the minimum number $n^{\min}$ for EPR. The minimum number of processing nodes that the task needs at time s to complete before its deadline is

$$n^{\min} = \left\lceil \frac{\sigma c_{ps}}{A + D - s - \sigma c_{ms}} \right\rceil$$

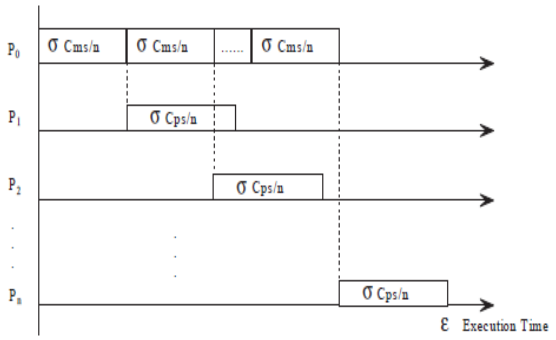


Figure 2: Time Diagram for EPR-Based Partitioning.

4.3 Task waiting queue (TWQ) algorithms

In these algorithms [10] *Mamat, Ying Lu, Jitender Deogun and Steve Goddard*, when a task arrives, the scheduler determines if it is feasible to schedule the new task without compromising the guarantees for previously admitted tasks. Only those tasks that pass this schedule ability test are allowed to enter the *task waiting queue* (TWQ). This decision module is referred to as the *admission controller*. When processing nodes become available, the *dispatcher* module partitions each task and dispatches subtasks to execute on processing nodes. In the Modules, admission controller and dispatcher, run on the head node. For existing divisible load scheduling algorithms [3, 4, 5, 6, 7], in order to perform the schedule ability test, the admission controller generates a new schedule for the newly arrived task and all tasks waiting in TWQ. If the schedule is feasible, the new task is accepted; otherwise, it is rejected. For these algorithms, the dispatcher acts as an execution agent, which simply implements the feasible schedule developed by the admission controller.

There are two factors that contribute to large overheads of these algorithms. First, to make an admission control decision, they reschedule tasks in TWQ. Second, they calculate in the admission controller the minimum number $n^{\min}$ of nodes required to meet a task's deadline so that it guarantees enough resources for each task. The later a task starts, the more nodes are needed to complete it before its deadline. Therefore, if a task is rescheduled to start at a different time, the $n^{\min}$ of the task may change and needs to be recomputed. This process of rescheduling and re-computing $n^{\min}$ of waiting tasks introduces a big overhead.

The dispatching algorithm [10] is rather straightforward. When a processing node and the head node become available, the dispatcher takes the first task τ (*A, σ, D*) in TWQ, partitions the task and sends a subtask of size $\hat{\sigma}$ to the node, where $\hat{\sigma} = \min\left(\frac{A + D - CurrentTime}{c_{ms} + c_{ps}}, \sigma\right)$. The remaining portion of the task τ (*A, σ-*$\hat{\sigma}$ *,D*) is left in TWQ. The dispatcher chooses a proper size $\hat{\sigma}$ to guarantee that the dispatched subtask completes no later than the task's absolute deadline *A* + *D*. Following the algorithm, all subtasks of a given task complete at the task absolute deadline, except for the last one, which may not be big enough to occupy the node until the task deadline. By dispatching the task as soon as the resources become available and letting the task occupy the node until the task deadline, the dispatcher allocates the minimum number of nodes to each task.

4.4 Case of Processor Ready Times

These algorithms can solve the real time divisible load by depending on the time which processing nodes are ready at it. This approach contains two kinds algorithms, algorithms when the processing nodes are equal ready time and algorithms when different ready time. The following subsections describe briefly the ideas of these algorithms.

4.4.1 Processors with Equal Ready Times

In [14, 15, 18], it is assumed that all the processors, upon which a particular job will be distributed by the head node, are available for that job over the entire time-interval between the instant that the head-node initiates data transfer to any one of these nodes, and the instant that it completes execution upon all the nodes. Under this model of processor availability, it is known that the completion time of a job on a given set of processing nodes is minimized if all the processing nodes complete their execution of the job at the same instant. This makes intuitive sense – if some processing node completes before the others for a given distribution of the job's workload, then a different distribution of the workload that transfers some of the assigned work from the remaining processing node to this one would have an earlier completion time. Figure 6 depicts the data transmission and execution time diagram when processors have equal ready times.

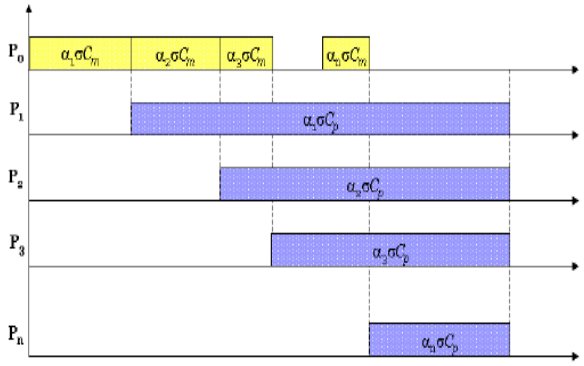


Figure 6: Data transmission and execution time diagram when processor have equal ready times

For a given job (A, σ, D) and a given number of processing nodes *n*, let $\sigma_i \times \alpha$ denote the amount of the load of the job that is assigned to the j^{th} processing node, $1 \le j \le n$. Since data-transmission occurs sequentially, the node $_i P$ can only receive data after the previous $(i-1)$ nodes have completed receiving their data. Hence, each p_i receives its data over the interval $\left[c_m \sum_{j=1}^{i-1} \alpha_j \sigma, c_m \sum_{j=1}^{i} \alpha_i \sigma \right)$

And therefore completes execution at time-instant $c_m \sum_{j=1}^{i} \alpha_j \sigma + c_p \alpha_i \sigma$ Then time execution time is assignment by equation (6) and to determine a minimum number of processors needed is computed from equation (6) by setting this completion time to the job's deadline (A+ D) in Equation (6), and making "*n*" — the number of processors — the variable. (Since the number of processors is necessary integral, it is actually the ceiling of this value that is the minimum number of processors.)

4.4.2 Processors with Different Ready Times Algorithm

In [16, 17], Lin et al. allow for the possibility that all the processors are not immediately available. To determine the completion time of a job upon a given number of processors in this more general setting, Lin et al.[16, 17, 21] adopt a *heuristic* approach that aims to partition a job so that the allocated processors could start at different times but finish computation (almost) simultaneously.

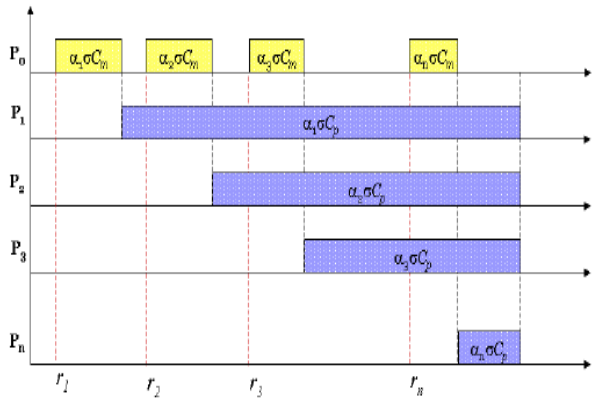
To achieve this, they first map the given homogenous cluster with different processor available times $r_1, r_2, ...r_n$ (with $\forall r_i \le r_{i+1}$) into a *heterogeneous* model where all *n* assigned nodes become available simultaneously at the time-instant r_n, but different processors may have different computing capacities. Intuitively speaking, the i^{th} processor has its computing capacity inflated to account for the reality that it is able to execute over the interval $[r_i, r_n)$ as well. Figure 7 depicts the data transmission and execution time diagram when processors have different ready times.

Figure 7: Data transmission and execution time diagram when processors have different ready times

In Lin et al [16, 17] , this heterogeneity is modeled by associating a different constant c_{ps_i} with each processor p_i, with the interpretation that it takes c_{ps_i} time to complete one unit of work on the processor p_i . The formula for determining c_{ps_i} , as given in Lin et al [16, 17] , is

$$c_{ps_i} = \frac{\xi(\sigma, n)}{\xi(\sigma, n) + r_n - r_i} \qquad (11)$$

Where $\xi(\sigma, n)$ denotes the completion time if all processors are immediately available in the original



(homogenous) cluster these c_{ps_i} values are used to derive formulas for computing the fractions of the workload that are to be allocated to each heterogeneous processor such that all processors complete at approximately the same time, and for computing this completion-time.

4.5 Least Cost Methods

G.K.Kamalam and Dr.V.Murali Bhaskaran [25] introduce a decentralized job scheduling algorithms which performs intra cluster and inter cluster (grid) job scheduling. They apply Divisible Load Theory (DLT) and Least Cost Method (LCM) to model the grid scheduling problem involving multiple resources within an intra cluster and inter cluster grid environment. The LCM method, the jobs are allocated to the resource with the least allocation cost [26]. The algorithm reduces the total processing time and the total cost and the resource utilization is more and the load is balanced across the grid environment.

Xin Liu et al [23,24] proposed another algorithm in which they tried to obtain minimum cost by perturbing the schedule of some tasks from minimum time solution. They proposed min-time algorithm to find the minimum completion time and the min-cost algorithm to find the minimum cost without considering the deadline constraint. Their proposed algorithm is a hybrid scheduling algorithm to minimize some of the tasks lying to the first of the list follow min time and the remaining tasks in the list follow min cost algorithm. This is called as perturbation degree. Their proposed algorithm stated that the task from the list is allowed to evaluate the minimum completion time and if it is greater than the deadline, then there is no possibility of getting feasible solution, if the minimum completion time is less than the deadline then binary search is used recursively for largest perturbation degree, such that the current or the next perturbation degree is smaller than the deadline. Now the cost and perturbation degree is obtained and returned as schedule with minimum cost and finished before deadline.

5 Metrics of real time divisible load for cluster scheduling

To measure the performance and distinguish between algorithms, different metrics are used. These metrics are used to measure the effects of parameters on these algorithms and it also merits between them.

The DC Ratio, task reject ratio, processing speed and number of nodes are the main performance metrics used by OPR, EPR and ready time processor algorithms. While task reject ratio, system utilization and scheduling overhead are used by TQW algorithms to measure their performance. The following subsections give brief descriptions of these metrics.

5.1 Effect of Task Reject Ratio

There is new metric *Task Reject Ratio* can use it to specify the real-time scheduling algorithm is better or not , which is define as the ratio of the number of tasks rejected by a real-time scheduling algorithm to the total number of tasks arriving at the cluster The smaller the *Task Reject Ratio*, the better the real-time scheduling algorithm. The *Task Reject Ratio* of the four algorithms: EDF-OPR-MN, EDF-EPR-MN, EDF-OPR-AN, and EDF-EPR-AN. Observe that EDF-OPR-MN always leads to a lower *Task Reject Ratio* than EDF-EPR-MN. Similarly, observe that EDF-OPR-AN always achieves a lower *Task Reject Ratio* than EDF-EPR-AN. These simulation results confirm the hypothesis [10] that it is advantageous to apply DLT in real-time, cluster-based scheduling algorithms. DLT provides an optimal task partitioning, which leads to minimum task execution times, and as a result the cluster can satisfy a larger number of task deadlines.

5.2 Effects of DCRatio

[1,2] There are another metric that effect on the real time algorithms that is DCRatio which is defined as the ratio of mean deadline to mean minimum execution time (cost), that is $\frac{\text{AvgD}}{\xi(\text{Avg}\sigma,\text{N})}$, where $\xi(\text{Avg}\sigma,\text{N})$ is the task execution time computed with Eq (6) assuming the task has an average data size $\text{Avg}\sigma$ and runs on all N processing nodes.

To study the effects of the DCRatio, on the real time algorithms of divisible load, observe that by increasing DCRatio, the performance of EDF-EPR-AN becomes closer to that of EDF-OPR-AN. This is because the higher the DCRatio, the looser the task relative deadlines are. Consequently, the worse execution times caused by a non-optimal partition, like EPR, will have less impact on the algorithms' performance. In particular, when DCRatio is extremely high (100), the two algorithms perform almost the same.

5.3 Effects of Processing Speed

By studying effects of processing speed, the algorithm with OPR [9,10] partitioning (EDF-OPR-MN) still outperforms the algorithm with EPR partitioning (EDF-EPR-MN). However, as the processing speed decreases, i.e., c_{ps} increases, the difference between the two algorithms becomes less and less significant. In particular, when the computation is extremely slow (c_{ps} = 10000), the curves for the two algorithms are almost overlapped, indicating non-differentiable *Task Reject Ratios*. Therefore, OPR and EPR will perform the same in this case. From the aforementioned intensive experiments, then the conclusion is no matter what the system parameters are, the algorithms with DLT-based partitioning (OPR) always perform better than the ones with the equal-sized partitioning heuristic (EPR). This shows that it is beneficial to apply divisible load theory in real-time, cluster-based scheduling.

5.4 All nodes N versus $n^{\min}$ Nodes

The performance of real time divisible load algorithm [9,10,20] difference in algorithms assigning all N nodes to every task (ALG-AN) v.s. those assigning the minimum number $n^{\min}$ of nodes needed to meet a task's deadline (ALG-MN). Where the relative performance of EDF-OPR-MN v.s. EDF-OPR-AN is noteworthy that in contrast to the results in [12] comparing MWF (-MN) and FIXED (-AN) algorithms, the initial data seem to indicate that EDF-OPR-AN outperforms EDF-OPR-MN most of the time. To gain insight into the performance results, Carry out rigorous analysis of a simplified scenario where a scheduling algorithm always assigns K nodes (K < N) to a periodic divisible task. This analysis sheds new light on possible scenarios where algorithms assigning $n^{\min}$ nodes (ALG-MN) perform better than those assigning all N nodes (ALG-AN).

5.5 Scheduling Overhead and cost

This metrics investigate the effect of scheduling overheads and deadline constrain. Theses algorithms try to minimize the overhead affected by the scheduling algorithms and meet the deadline constrain.

6. Conclusion

In this paper, the real-time divisible load distribution problem in computational grid is investigated. We try to present the progress and developing efforts to determine the best mechanisms, policies and analysis to use in these systems. Different matrices and constrains can be compromised by building systems using approaches that lack the necessary theoretical underpinnings. Ultimately, computational grid will be used in high integrity real-time systems, and consequently, timing failures could affect safety. The paper study different scheduling algorithms; scheduling policies, and hybrid algorithms. Comparisons of fewer algorithms with various factors influencing Grid system are explained. An investigation on various factors that influence the scheduling in grid has been made and shown in this paper. This is an effort made to find the silver lining in the dark clouds which could paint an idea about the scheduling policies applied to the real-time divisible load problem in computational environment.

References

[1] Suriayati bt chuprat," The deadline-based scheduling of Divisible Real-Time Workloads on Multiprocessor

Platforms", PHD degree of Doctor of Philosophy (Mathematics), Faculty of science, university technology Malaysia 2009.
[2] Computer Science and Engineering, Department of Computer Science and Engineering: "Real-Time Divisible Load Scheduling For Cluster Computing", Theses Dissertations, and Student Research University of Nebraska - Lincoln Year 2011.
[3] S. Chuprat and S. Baruah. " Scheduling divisible real-time loads on clusters with varying processor start times" In 14th IEEE International Conference on Embedded and Real-Time
Computing Systems and Applications (RTCSA '08),pages 15–24, Aug 2008.
[4] S. Chuprat, S. Salleh, and S. Baruah. "Evaluation of a linear programming approach towards scheduling divisible real-time Loads", In International Symposium on Information Technology, pages 1–8, Aug 2008.
[5] W. Y. 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] X. Lin, Y. Lu, J. Deogun, and S. Goddard, "Real-time divisible load scheduling with different processor available times", In Proceedings of the 2007 International Conference on Parallel Processing (ICPP 2007).
[7] X. Lin, Y. Lu, J. Deogun, and S. Goddard.," Real-time divisible load scheduling for cluster computing" In Proceedings of the 13th IEEE Real-Time and Embedded Technology and Application Symposium pages 303–314, Bellevue, WA, April 2007.
[8] A. Mamat, Y. Lu, J. Deogun, and S. Goddard,"Real-time divisible load scheduling with advance reservations", In 20th Euromicro Conference on Real-Time Systems pages 37–46, July 2008.
[9] Anwar Mamat, Ying Lu, Jitender Deogun, Steve Goddard, "An Efficient Algorithm for Real-Time Divisible Load Scheduling", Department of Computer Science and Engineering University of Nebraska – Lincoln Lincoln NE 68588 {anwar, ylu, deogun, goddard}@cse.unl.edu
[10] B. Veeravalli, D. Ghose, and T. G. Robertazzi., "Divisible load theory: A new paradigm for load scheduling in distributed systems Cluster Computing", 6(1):7–17, 2003.
[11] D. Swanson. Personal communication. Director, UNL Research Computing Facility (RCF) and
UNL CMS Tier-2 Site, August 2005.
[12] D. Isovic and G. Fohler. "Efficient scheduling of sporadic periodic, and periodic tasks with complex constraints", In Proc. of 21st IEEE Real-Time Systems Symposium,pages 207–216, Orlando, FL, November 2000.
[13] B. Veeravalli, D. Ghose, and T. G. Robertazzi, "Divisible load theory: A new paradigm for load scheduling in distributed systems Cluster Computing", 6(1):7–17, 2003.
[14] Lin, X., Lu, Y., Deogun, J. and Goddard, S., " Real-time divisible load scheduling for cluster computing", Technical Report UNL-CSE-2006-0016 (2006a), Department of Computer Science and Engineering, The University of Nebraska at Lincoln.
[15] Lin, X., Lu, Y., Deogun, J. and Goddard, S., "Real-time divisible load scheduling for clusters. Proceedings of the Real-Time Systems Symposium",(2006b).– Work-In-Progress Session pages 9–12, Rio de Janerio, December.
[16] Lin, X., Lu, Y., Deogun, J. and Goddard, S., " Real-Time Divisible Load Scheduling with
Different Processor Available Times", Proceedings of International Conference on Parallel Processing (ICPP), (2007b), Xian, China, September.
[17] Lin, X., Lu, Y., Deogun, J. and Goddard, S., " Enhanced Real-Time Divisible Load Scheduling with Different Processor Available Times", Proceedings of 14th International Conference On High Performance Computing (HiPC), (2007c) Goa, India, December.
[18] Compact Muon Solenoid (CMS) Experiment for the Large Hadron Col-lider at CERN (European Lab for Particle Physics), Cms web page, http://cmsinfo.cern.ch/Welcome.html/.
[19] ATLAS (A Toroidal LHC Apparatus) Experiment, CERN (European Lab for Particle Physics), Atlas web page, http://atlas.ch/.
[20] Xuan Lin, Anwar Mamat, Ying Lu_, Jitender Deogun, Steve Goddard," Real-time scheduling of divisible loads in cluster computing environments" , J. Parallel Distrib. Comput. 70 (2010) 296_308
[21] Kijeung Choi 1, Thomas G. Robertazzi , "An Exhaustive Approach to Release Time Aware Divisible Load Scheduling", (IJIDCS) International Journal on Internet and Distributed Computing Systems. Vol: 1 No: 2, 2011.
[22] G. Murugesan and C. Chellappan, "An Economic Allocation of Resources for Divisible Workloads in Grid Computing Paradigm", European Journal of Scientific Research, ISSN 1450-216X Vol.65 No.3 (2011), pp. 434-443 © EuroJournals Publishing, Inc. 2011
[23] Xin Liu, Chunming Qiao, Wei Wei, Xiang Yu,Ting Wang, Weisheng Hu, Wei Guo, and Min-You Wu, "Task Scheduling and Lightpath Establishment in Optical Grids", Journal Of Light Wave Technology, 2009, p 1796-1805.
[24] Dr. D.I. George Amalarethinam1, P. Muthulakshmi , "An Overview of the Scheduling Policies and Algorithms in Grid Computing" , International Journal of Research and Reviews in Computer Science (IJRRCS) Vol. 2, No. 2, April 2011.
[25] Syed Nasir Mehmood Shah, Ahmad Kamil Bin Mahmood, and Alan Oxley 2010, "Hybrid Resource Allocation for Grid Comp uting", in Proceed in gs of the IEEE Second International Conference on Computer Research and Development, 426 – 431.
[26] G.K.Kamalam,and Dr.V.Murali Bhaskaran, "An Effective Approach to Job Scheduling in Decentralized Grid Environment", International Journal of Computer Applications (0975 – 8887) Volume 24– No.1, June 2011.