University of Nebraska - Lincoln
DigitalCommons@University of Nebraska - Lincoln

CSE Technical reports | Computer Science and Engineering, Department of

2-27-2007

Real-Time Divisible Load Scheduling with Different Processor Available Times

Xuan Lin
University of Nebraska - Lincoln

Steve Goddard
University of Nebraska - Lincoln, goddard@cse.unl.edu

Ying Lu
University of Nebraska - Lincoln, ylu4@unl.edu

Jitender Deogun
University of Nebraska - Lincoln, jdeogun1@unl.edu

Follow this and additional works at: http://digitalcommons.unl.edu/csetechreports

Part of the Computer Sciences Commons

Lin, Xuan; Goddard, Steve; Lu, Ying; and Deogun, Jitender, "Real-Time Divisible Load Scheduling with Different Processor Available Times" (2007). *CSE Technical reports.* Paper 41.
http://digitalcommons.unl.edu/csetechreports/41

This Article is brought to you for free and open access by the Computer Science and Engineering, Department of at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in CSE Technical reports by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.

University of Nebraska–Lincoln, Computer Science and Engineering
Technical Report TR-UNL-CSE-2007-0013
Issued Feb. 27, 2007

Technical Report UNL-CSE-2007-0013
Real-Time Divisible Load Scheduling with Different Processor Available Times

Xuan Lin, Ying Lu, Jitender Deogun, Steve Goddard
Department of Computer Science and Engineering
University of Nebraska - Lincoln
Lincoln, NE 68588
{*lxuan, ylu, deogun, goddard*}*@cse.unl.edu*

Abstract

Providing QoS and performance guarantees to arbitrarily divisible loads has become a significant problem for many cluster-based research computing facilities. While progress is being made in scheduling arbitrarily divisible loads, some of proposed approaches may cause Inserted Idle Times (IITs) that are detrimental to system performance. In this paper we propose a new approach that utilizes IITs and thus enhances the system performance. The novelty of our approach is that, to simplify the analysis, a homogenous system with IITs is transformed to an equivalent heterogeneous system, and that our algorithms can schedule real-time divisible loads with different processor available times. Intensive simulations show that the new approach outperforms the previous approach in all configurations. We also compare the performance of our algorithm to the current practice of manually splitting workloads by users. Simulation results validate the advantages of our approach.

Keyword: Real-Time Scheduling; Inserted Idle Time; Cluster Computing; Divisible Load.

1 Introduction

Arbitrarily divisible or embarrassingly parallel workloads, can be partitioned into an arbitrarily large number of independent load fractions, and are quite common in bioinformatics as well as high energy and particle physics. For example, the CMS (Compact Muon Solenoid) [14] and ATLAS (AToroidal LHC Apparatus) [6] projects, associated with the Large Hadron Collider (LHC) at CERN (European Laboratory for Particle Physics), execute cluster-based applications with arbitrarily divisible loads.

Development of commodity-based clusters has recently gained considerable momentum. By linking a large number of computers together, a cluster provides a cost-effective facility for solving complex problems. In a large-scale cluster, the resource management system (RMS), which provides real-time guarantees or QoS, is central to its operation.

As a result, the real-time scheduling of arbitrarily divisible loads is becoming a significant problem for cluster-based research computing facilities like the U.S. CMS Tier-2 sites [33]. One of the management goals at the University of Nebraska-Lincoln (UNL) Research Computing Facility (RCF) is to provide a multi-tiered QoS scheduling framework in which applications "*pay*" according to the response time requested for each job [33]. Due to the increasing importance [28], a few efforts [18, 20, 22] have been made in real-time divisible load scheduling, with significant initial progress in important theories and novel approaches.

However, an important classic problem of scheduling parallel jobs has not yet been adequately addressed for real-time divisible load scheduling. For executing a parallel job, if a sufficient number of processors are available then the processors are allocated and the job is started. But if the required number of processors are not available, the job waits for some currently running jobs to finish and free additional processors. This essentially causes a waste of processing power as some processors are idle when the system is waiting for enough processors to become available to start the waiting job. This drawback is a system inefficiency that we refer to as the Inserted Idle Times (IITs) problem. To alleviate this limitation, backfilling algorithms [21, 24, 29] have been proposed, where small jobs could be moved ahead and run on processors that would otherwise remain idle. Leveraging characteristics of arbitrarily divisible loads, we propose in this paper a new real-time divisible load scheduling approach that utilizes IITs. The novelty that distinguishes our work from conventional approaches is that *our algorithms can schedule real-time divisible loads with different processor available times*. Our approach is complementary to the backfilling mechanism.

Divisible load theory (DLT) provides insight into distribution strategies for arbitrarily divisible computations and has been demonstrated to lead to significantly better approaches for real-time divisible load scheduling [22]. Two contributions are made in this paper. First, we *cast a homogenous cluster with different processor available times to a heterogeneous cluster model*. A DLT heterogeneous model is then applied to guide task partitioning, to derive a task execution time function and to approximate the minimum number of processors required to meet a task deadline. Second, we *prove that executing the partitioned subtasks in the homogenous cluster at different processor available times leads to completion times no later than the estimates*. This result is then applied to develop a new divisible load scheduling algorithm that uses IITs and provides real-time guarantees. Intensive simulation results show that the new approach outperforms the previous approach [22] where IITs are not utilized. We also compare the algorithm with a current practice where users manually split their workloads. Results demonstrate the obvious advantages of the proposed approach.

The remainder of this paper is organized as follows. Related work is presented in Section 2. We describe both task and system models in Section 3. In Section 4, real-time scheduling algorithms inves-

tigated in this paper are discussed. We evaluate the performance of algorithms in Section 5 and conclude the paper in Section 6.

2 Related Work

Utility-driven cluster computing has been well researched [31, 35] to improve the utility delivered to users. Proposed cluster RMSs [3, 12] have addressed the scheduling of both sequential and parallel loads. The goal of those schemes is similar to ours: *to harness the power of resources based on user objectives*.

The scheduling models investigated for real-time distributed systems most often (e.g., [1, 5, 17, 19, 26, 27]) assume periodic or aperiodic sequential jobs that must be allocated to a single resource and executed by their deadlines. With the evolution of cluster computing, researchers have begun to investigate real-time scheduling of parallel applications [2, 4, 16, 25, 36]. However, most of these studies assume the existence of some form of task graph to describe communication and precedence relations between computational units called subtasks (i.e., nodes in the task graph). Despite the increasing importance of arbitrarily divisible applications [28], to the best of our knowledge, only a few researchers [18, 20, 22] have investigated the real-time scheduling of arbitrarily divisible loads.

The most closely related work to ours is the scheduling of "scalable tasks" [20] or "moldable jobs" [7, 13, 18, 30, 32], where only very few of them [18, 20] have considered QoS support. In [22] we investigated real-time cluster-based divisible load scheduling and proposed several algorithms for homogenous clusters. Following our previous work, in this paper, we develop a real-time scheduling approach that utilizes Inserted Idle Times. A mechanism to utilize processor idle-times, also called fragments, was investigated in [20]. However, that approach is different from ours. They try to utilize more processing power by assigning a task to a larger number of nodes. Complementary to their approach, our algorithm enables a task to utilize a processor as soon as it becomes available. Results shown in [20] depict that the performance improvement of the approach in [20] is negligible. In contrast, our algorithm leads to significantly better performance.

DLT provides an in-depth study of distribution strategies for arbitrarily divisible loads [10, 28, 34]. The goal of DLT is to exploit parallelism in computational data so that the workload can be partitioned and assigned to several processors such that execution completes in the shortest possible time [10]. In our previous work [22], we demonstrated that the application of DLT leads to significantly better approaches for real-time divisible load scheduling. Encouraged by its performance benefits, we again apply DLT to develop new algorithms. Specifically, a DLT heterogeneous model is applied in the partitioning of applications, such as CMS [14] and ATLAS [6], that execute on a large homogenous cluster.

3 Task and System Models

In this paper, we adopt the same task and system models as our previous work [22]. For completeness, we briefly present these below.

Task Model. We assume a real-time aperiodic task model in which each aperiodic task T_i consists of a single invocation specified by the tuple (A_i, σ_i, D_i), where A_i is the task arrival time, σ_i is the total data size of the task, and D_i is its relative deadline. The task absolute deadline is given by $A_i + D_i$. Section 4.1 presents, in detail, how 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.

System Model. A cluster consists of a head node, denoted by P_0, connected via a switch to N processing nodes, denoted by P_1, P_2, $\ldots$, P_N. We assume 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. We assume that data transmission does not occur in parallel, although it is straightforward to generalize our model and include the case where some pipelining of communication may occur. For the arbitrarily divisible loads, tasks 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 [34]. 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. For many applications the output data is just a short message and is negligible, particularly considering the very large size of the input data. Therefore, in this paper we only model the transfer of application input data but not that of output data. The extension to consider the transfer of output data using DLT is straightforward.

The following notations, partially adopted from [34], are used in this paper,

- $T = (A, \sigma, D)$: A divisible task, where A is arrival time, σ is data size, and D is 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 jth node, i.e., $\alpha_j \sigma$, is the amount of data that is to be transmitted to the jth node for processing, $0 < \alpha_j \leq 1$ and $\Sigma_{j=1}^{n} \alpha_j = 1$;

- Cms: Cost of transmitting a unit workload;
- Cps: Cost of processing a unit workload.

4 Algorithms

This section presents real-time divisible load scheduling algorithms that utilize Inserted Idle Times (IITs) in a cluster. Many parallel job scheduling algorithms [21, 22] face the IITs problem. It occurs when the number of processors available is less than that required by the next job. In that case, the job has to wait until enough processors become available, which leads to a waste of processing power as some processors are idle, the so-called Inserted Idle Times (IITs) problem. Backfilling [21, 24, 29] is an approach proposed in the literature to alleviate this problem. It is a general approach applicable to all types of parallel jobs — whether modularly divisible or arbitrarily divisible.

An arbitrarily divisible load, however, has a very unique property, that is, it can be arbitrarily partitioned into a large number of independent subtasks of arbitrary size. Thus, the subtasks can be scheduled flexibly and independently. Exploiting this property of arbitrarily divisible loads, we propose new algorithms that schedule divisible loads with different processor available times and utilize IITs in a cluster.

In [22], we encapsulated the logic of a real-time divisible load scheduling algorithm in three modules. The first module determines the task execution order, which could be based on policies like FIFO (first in first out) or EDF (earliest deadline first). The second task partitioning module chooses a strategy to divide loads while the third module decides the node assignment for each task. In this paper, to utilize IITs we focus on the second module, that is, designing a new task partitioning module for real-time divisible load scheduling.

To allocate resources to meet a divisible task deadline, a scheduling algorithm must know the minimum amount of resources required by the task, which is determined by the task data size and the partitioning method applied. For a homogenous cluster, that amounts to computing the task execution time and the minimum n^{min} number of nodes required. Our previous work [22] addressed this issue when processors are simultaneously allocated to a task (Figure 1a). To tackle the IITs problem, our new approach is designed to handle the scenario where processors can be allocated to a task at different times (Figure 1b). This makes task partitioning, execution time analysis, and derivation of n^{min} difficult.

4.1 Task Partitioning and Analysis

We investigate two partitioning methods: *DLT-Based Partitioning* (Section 4.1.1), and *User-Split Partitioning* (Section 4.1.2). The first method is based on divisible load theory (DLT), which states the optimal execution time is obtained when all nodes allocated to a task finish their computation at the

same time [34]. For comparison, we propose the User-Split Partitioning method, based on a common practice of the user dividing a task into n equal-sized subtasks when n nodes are requested for the task.

4.1.1 DLT-Based Partitioning

Although divisible load theory has been extensively studied in recent years, most of the developed models assume that all nodes are allocated at the same time. To the best of our knowledge, very little work [9, 8, 11] addresses the load partitioning problem with different processor available times. However, their solutions are not applicable to typical clusters because they assume that when a processor is computing the current task, the system could deliver the data of the next task to it. Without special hardware and software support this assumption does not hold in common cluster environments.

Following the principle of DLT, our heuristic approach aims to partition a task so that the allocated processors could start at different times but finish computation almost simultaneously.

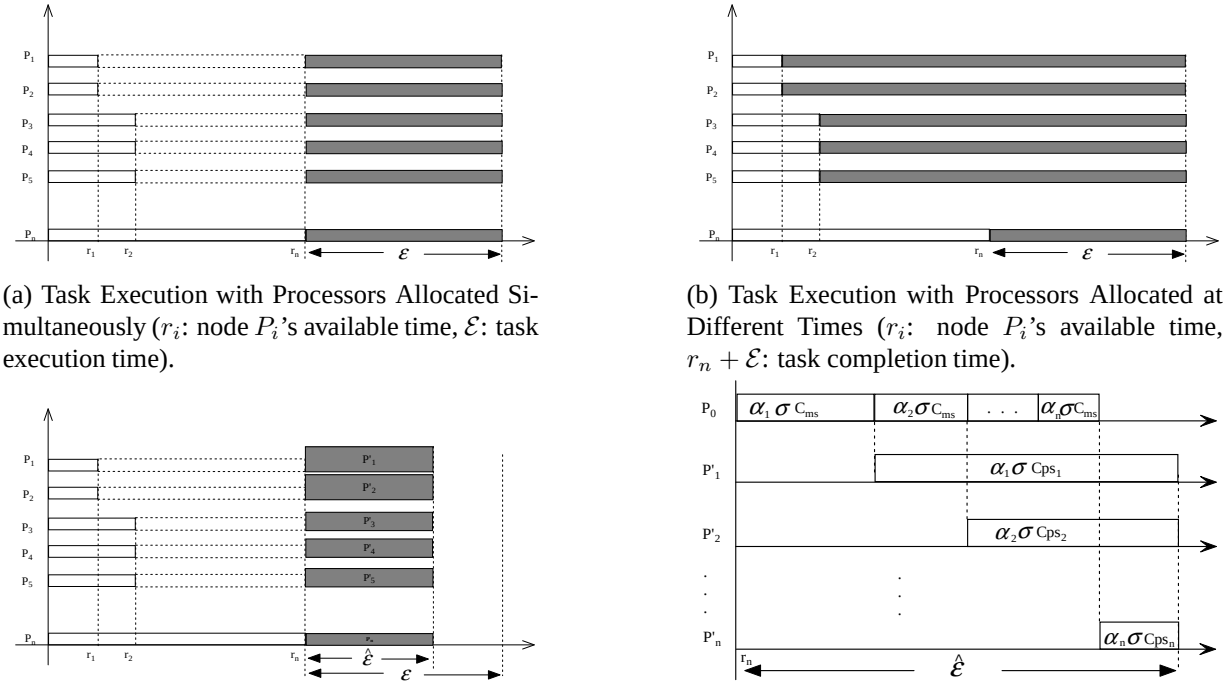
To achieve the aforementioned objective, we first cast a homogenous cluster with different processor available times into a heterogeneous model where all assigned nodes are allocated simultaneously (**A: Heterogeneous Model Construction**). Constructing such a model enables us to apply DLT to guide task partitioning, execution time analysis, and n^{min} nodes derivation (**B: Applying DLT**). Then we prove if we partition the task following the model and execute the subtasks in the homogenous cluster, the actual task completion time is no later than its estimate (**C: Analysis of Completion Time Estimate**) — a necessary condition to guarantee the correctness of our real-time scheduling algorithm (Section 4.2).

A: Heterogeneous Model Construction

Given n homogenous processors to start the execution of a divisible task T at different available times, we transform them to a model of n heterogeneous nodes allocated simultaneously to the task.

Let $P_1, P_2, \ldots, P_n$ denote the n homogenous processors. Assume node P_i could start processing task T at time r_i, for $i = 1, 2, \ldots n$. We call r_i the available time of P_i. It is either the time P_i is released by a previous task or the time task T arrives, whichever is latest. The n nodes are ordered by their available times: P_1 is the earliest at time r_1 and P_n the latest at time r_n (Figure 1b). Next we construct a model of n heterogeneous nodes with the same allocation time r_n (Figure 1c).

As demonstrated by Figures 1b and 1c, the earlier the homogenous node becomes available, the more powerful is its corresponding node in the heterogeneous model. In Figure 1c, a different height of a dark rectangular bar is used to represent a different node processing power. The greater the height of the bar, the more powerful the node. In the new model (Figure 1c) all n nodes are considered to be allocated at the same time r_n. The effect of the Inserted Idle Time (IIT) of a homogenous node, $r_n - r_i$, is accounted for by assuming its corresponding heterogeneous node $P_i^{'}$ has a higher processing power. Specifically, in the constructed model, while the link speed is considered the same as the original cluster, the node



(a) Task Execution with Processors Allocated Simultaneously (r_i: node P_i's available time, $\mathcal{E}$: task execution time).

(b) Task Execution with Processors Allocated at Different Times (r_i: node P_i's available time, $r_n + \mathcal{E}$: task completion time).

(c) Heterogeneous Cluster Model (r_i: node P_i's available time, $\mathcal{E}$: task execution time without utilizing IITs, $\hat{\mathcal{E}}$: task execution time in the heterogeneous model).

(d) Time Diagram for the Heterogeneous Model (r_n: the time all n nodes become available for task T, $r_n + \hat{\mathcal{E}}$: task completion time).

Figure 1: Heterogeneous Model Construction.

computational power is increased based on the length of the IIT and the task execution time when no IIT is utilized. Let $\mathcal{E}$ (Figure 1a) denote the task execution time when no IIT is utilized (see [22] for $\mathcal{E}$'s derivation). Cps_i represents the unit processing cost on node $P_i^{'}$ and Cms_i denotes the unit transmission cost. Then, for the heterogeneous model, we have the following,

$$Cps_i = \frac{\mathcal{E}}{\mathcal{E} + r_n - r_i} Cps \quad (1)$$

$$Cms_i = Cms \quad (2)$$

Cps_i increases as i increases. Since the greater the Cps_i the slower the node, $P_1^{'}, P_2^{'}, \ldots P_n^{'}$ have decreasing processing power in the heterogeneous model.

B: DLT-Based Analysis

We now explain how to apply a DLT heterogeneous model for task partitioning, execution time analysis, and n^{min} derivation.

Task Partitioning and Execution Time Analysis. Figure 1d shows an example task execution time diagram following the optimum partition rule of DLT [34] when n heterogeneous nodes are allocated to process task $T = (A, \sigma, D)$ at the same time r_n. Let $\hat{\mathcal{E}}$ denote the task execution time. It is a function of

σ and n. That is,

$$\begin{aligned}
\hat{\mathcal{E}}(\sigma, n) &= \alpha_1 \sigma Cms + \alpha_1 \sigma Cps_1 \\
&= (\alpha_1 + \alpha_2)\sigma Cms + \alpha_2 \sigma Cps_2 \\
&= (\alpha_1 + \alpha_2 + \alpha_3)\sigma Cms + \alpha_3 \sigma Cps_3 \\
&= ... \\
&= \sum_{i=1}^{n} \alpha_i \sigma Cms + \alpha_n \sigma Cps_n \qquad (3)
\end{aligned}$$

$$\begin{aligned}
\text{Thus we have,} \qquad \alpha_2 &= \frac{Cps_1}{Cms + Cps_2}\alpha_1 \\
\alpha_3 &= \frac{Cps_2}{Cms + Cps_3}\alpha_2 \\
&... \\
\alpha_n &= \frac{Cps_{n-1}}{Cms + Cps_n}\alpha_{n-1} \\
\text{Let} \qquad X_i &= \frac{Cps_{i-1}}{Cms + Cps_i}, \text{ for } i = 2, 3 \ldots, n \\
\text{Then} \qquad \alpha_i &= X_i \alpha_{i-1} = X_i X_{i-1} \alpha_{i-2} = \cdots \\
&= \prod_{j=2}^{i} X_j \alpha_1, \text{ for } i = 2, 3 \ldots, n
\end{aligned}$$

Since $\sum_{i=1}^{n} \alpha_i = 1$, we have, $(1 + \sum_{i=2}^{n} \prod_{j=2}^{i} X_j)\alpha_1 = 1$
Thus, $\alpha_1 = \frac{1}{1+\sum_{i=2}^{n} \prod_{j=2}^{i} X_j}$. Then the data chunk allocated to node $P_1^{'}$ is,

$$\sigma_1 = \alpha_1 \sigma = \frac{\sigma}{1 + \sum_{i=2}^{n} \prod_{j=2}^{i} X_j} \qquad (4)$$

And the data chunks allocated to the other nodes are,

$$\sigma_i = \alpha_i \sigma = \prod_{j=2}^{i} X_j \alpha_1 \sigma = \frac{\prod_{j=2}^{i} X_j \sigma}{1 + \sum_{i=2}^{n} \prod_{j=2}^{i} X_j}, \text{ for } i = 2, 3 \ldots, n \qquad (5)$$

Task execution time can therefore be calculated as follows,

$$\begin{aligned}
\hat{\mathcal{E}}(\sigma, n) &= \sum_{i=1}^{n} \alpha_i \sigma Cms + \alpha_n \sigma Cps_n = \sigma Cms + \alpha_n \sigma Cps \\
&= \sigma Cms + \prod_{j=2}^{n} X_j \alpha_1 \sigma Cps = \sigma Cms + \frac{\prod_{j=2}^{n} X_j}{1 + \sum_{i=2}^{n} \prod_{j=2}^{i} X_j} \sigma Cps \qquad (6)
\end{aligned}$$

Derivation of an Upper-Bound for n^{min}. Given the task execution time function $\hat{\mathcal{E}}(\sigma, n)$, we can calculate an upper-bound for the minimum number n^{min} of nodes required to meet the task deadline.

Let $C(n)$ denote the task completion time function. Assume task $T = (A, \sigma, D)$ has a start time r_n. Then $C(n) = r_n + \hat{\mathcal{E}}(\sigma, n)$, that is,

$$C(n) = r_n + \sigma Cms + \frac{\prod_{j=2}^{n} X_j}{1 + \sum_{i=2}^{n} \prod_{j=2}^{i} X_j} \sigma Cps \tag{7}$$

To meet the task deadline means constraint $C(n) \leq A + D$ must be satisfied. It follows that,

$$r_n + \sigma Cms + \frac{\prod_{j=2}^{n} X_j}{1 + \sum_{i=2}^{n} \prod_{j=2}^{i} X_j} \sigma Cps \leq A + D$$

Since it is difficult to solve the above inequality to get the exact minimum number n^{min}, we derive a new number $\tilde{n}^{min}$ that is an upper bound for n^{min} ($\tilde{n}^{min} \geq n^{min}$). This way, if the scheduler allocates at least $\tilde{n}^{min}$ processors to the task, the deadline will be guaranteed.

From Eq. (1) and the fact $r_n \geq r_i \geq r_{i-1}$, we know $Cps_{i-1} \leq Cps_i$ and $Cps_i \leq Cps$. Therefore we have,

$$X_i = \frac{Cps_{i-1}}{Cms + Cps_i} \leq \frac{Cps}{Cms + Cps}, \text{ for } i = 2, 3 \ldots, n$$

Let

$$\beta = \frac{Cps}{Cms + Cps} \tag{8}$$

From $X_i \leq \beta$ and Eq. (6), we can prove the following for the task execution time,

$$\hat{\mathcal{E}}(\sigma, n) \leq \sigma Cms + \beta^{n-1} \frac{1-\beta}{1-\beta^n} \sigma Cps$$

After algebraic simplification, we have,

$$\begin{aligned} \hat{\mathcal{E}}(\sigma, n) &\leq \frac{1-\beta}{1-\beta^n} \sigma (C_{ms} + C_{ps}), \text{ that is,} \\ \hat{\mathcal{E}}(\sigma, n) &\leq \mathcal{E}(\sigma, n) \end{aligned} \tag{9}$$

where $\mathcal{E}(\sigma, n)$ (see [22]) is the task execution time when no IITs are utilized. This leads to,

$$C(n) = r_n + \hat{\mathcal{E}}(\sigma, n) \leq r_n + \frac{1-\beta}{1-\beta^n} \sigma (C_{ms} + C_{ps}) \tag{10}$$

Thus by solving,

$$r_n + \frac{1-\beta}{1-\beta^n} \sigma (C_{ms} + C_{ps}) \leq A + D \tag{11}$$

we will get an upper bound number $\tilde{n}^{min}$ that also satisfies constraint $C(n) \leq A + D$. Eq. (11) implies,

$$\frac{1-\beta}{1-\beta^n} \sigma (C_{ms} + C_{ps}) \leq A + D - r_n \tag{12}$$

Since $1-\beta^n > 0$, we multiply both sides of (12) by $1-\beta^n$and get,

$$(1-\beta)\sigma(C_{ms}+C_{ps}) \leq (1-\beta^n)(A+D-r_n) \tag{13}$$

If $A+D-r_n \leq 0$, the task will miss its deadline no matter how many nodes we assign to it and how we partition it. Such a task will be rejected[1] since it fails the schedulability test of our algorithm (see Section 4.2). Thus $A+D-r_n > 0$, and by dividing both sides of (13) by $(A+D-r_n)$ we have,

$$\begin{aligned}(1-\beta^n) &\geq \frac{(1-\beta)\sigma(C_{ms}+C_{ps})}{A+D-r_n}, \quad \text{that is,}\\ \beta^n &\leq 1-\frac{(1-\beta)\sigma(C_{ms}+C_{ps})}{A+D-r_n} = 1-\frac{\sigma C_{ms}}{A+D-r_n}\end{aligned}$$

$$\text{Let} \qquad \gamma = 1-\frac{\sigma C_{ms}}{A+D-r_n} \tag{14}$$

we have $\beta^n \leq \gamma$. If $\gamma \leq 0$, starting task T at time r_n will not leave enough time even for its data transmission. Therefore the task will be rejected as well. Thus, $\gamma > 0$. Since $0 < \beta < 1$, it follows that $n \geq \frac{\ln\gamma}{\ln\beta}$. The assigned number n of nodes should be an integer. We have $n \geq \lceil\frac{\ln\gamma}{\ln\beta}\rceil$. Therefore $\tilde{n}^{min} = \lceil\frac{\ln\gamma}{\ln\beta}\rceil$, where γ is defined in Eq. (14) and β in Eq. (8). As long as the scheduler allocates at least $\tilde{n}^{min}$ nodes to task T at time r_n, the task deadline would be guaranteed.

C: Analysis of Completion Time Estimate

In the above, we have explained how our heuristic partitions a divisible task (Eq. (4) & (5)) following a DLT heterogeneous model. In addition, the task completion time $r_n + \hat{\mathcal{E}}(\sigma, n)$ is derived.

As explained, the heterogeneous model is constructed to guide task partitioning, execution time analysis and $\tilde{n}^{min}$ derivation. In reality, these ensuing subtasks are assigned and executed at the homogenous cluster. The derived completion time $r_n + \hat{\mathcal{E}}(\sigma, n)$ is therefore an estimate of the actual value.

Next we prove that the actual completion time is no worse than its estimate. Thus, if we use estimated completion times to schedule real-time divisible tasks, we can guarantee their temporal correctness.

Assertion 1 $\alpha_i < \alpha_1,$ *for* $i = 2, 3 \ldots, n$

Proof From $Cms > 0$ and $Cps_{i-1} \leq Cps_i$, we know,

$$\begin{aligned}X_i &= \frac{Cps_{i-1}}{Cms + Cps_i} < 1, \ i = 2, 3 \ldots, n.\\ \text{Thus,} \qquad \alpha_i &= \prod_{j=2}^{i} X_j \alpha_1 < \alpha_1, \ \text{for } i = 2, 3 \ldots, n.\end{aligned}$$

[1]Rejection in the cluster environment means that the system administrator (or a program proxy) will negotiate with the client for a feasible task deadline and the job will be rescheduled with modified parameters.

Lemma 2 $\alpha_i < \frac{Cps_1}{Cps_i}\alpha_1,$ *for* $i = 2, 3 \ldots, n$

Proof

$$\begin{aligned}\alpha_i &= \prod_{j=2}^{i} X_j \alpha_1 = \frac{\prod_{j=1}^{i-1} Cps_j}{\prod_{j=2}^{i}(Cms + Cps_j)}\alpha_1 \\ &< \frac{\prod_{j=1}^{i-1} Cps_j}{\prod_{j=2}^{i} Cps_j}\alpha_1 = \frac{Cps_1}{Cps_i}\alpha_1\end{aligned}$$

Assertion 3 $r_n - r_i \geq \frac{Cps}{Cps_i}\hat{\mathcal{E}} - \hat{\mathcal{E}}$

Proof From Eq. (1), we have $r_n - r_i = \frac{Cps}{Cps_i}\mathcal{E} - \mathcal{E}$ and $\frac{Cps}{Cps_i} \geq 1$. In addition, Eq. (9) says $\mathcal{E} \geq \hat{\mathcal{E}}$. Therefore,

$$r_n - r_i = \frac{Cps}{Cps_i}\mathcal{E} - \mathcal{E} \geq \frac{Cps}{Cps_i}\hat{\mathcal{E}} - \hat{\mathcal{E}}$$

Theorem 4 *The actual time for node* $P_i,\ \forall i \in \{1, 2, \cdots, n\}$*, to finish its computation is no later than the estimated task completion time.*

Proof According to the constructed model (Figure 1c), the estimated completion time t_est for all nodes are the same, $t_est = \hat{\mathcal{E}} + r_n$. While the actual completion time for node P_i is $t_act_i = \alpha_i\sigma(Cms + Cps) + r_i + \lambda_i$, where the first term is the communication and computation times for data assigned to P_i and the second term is the node available time. Since the cluster sequentially sends data chunks to corresponding nodes, the data transmission for node P_i cannot start until the cluster has finished transmitting data to nodes $P_1, P_2, \cdots, P_{i-1}$. The delay caused is represented by the third term of the above equation. From Figure 1d, we can see that the longest delay is caused when node P_i is available at the same time as node P_1. If we use $\tilde{\lambda}_i$ to denote this upper-bound for λ_i ($\lambda_i \leq \tilde{\lambda}_i$), we have,

$$\tilde{\lambda}_i = \sum_{j=1}^{i-1} \alpha_j \sigma Cms$$

It implies the following upper-bound $t_\tilde{act}_i$ for the actual completion time t_act_i,

$$t_\tilde{act}_i = \sum_{j=1}^{i} \alpha_j \sigma Cms + \alpha_i \sigma Cps + r_i$$

Thus, we have,

$$\begin{aligned}t_est - t_\tilde{act}_i &= \hat{\mathcal{E}} + r_n - (\sum_{j=1}^{i} \alpha_j \sigma Cms + \alpha_i \sigma Cps + r_i) \\ &= (r_n - r_i) + \hat{\mathcal{E}} - (\sum_{j=1}^{i} \alpha_j \sigma Cms + \alpha_i \sigma Cps)\end{aligned}$$

By applying Assertion 3, we get,

$$\begin{aligned} t_est - t_\tilde{act}_i &\geq (\frac{Cps}{Cps_i}\hat{\mathcal{E}} - \hat{\mathcal{E}}) + \hat{\mathcal{E}} - (\sum_{j=1}^{i} \alpha_j \sigma Cms + \alpha_i \sigma Cps) \\ &= \frac{Cps}{Cps_i}\hat{\mathcal{E}} - (\sum_{j=1}^{i} \alpha_j \sigma Cms + \alpha_i \sigma Cps) \end{aligned}$$

Since $\hat{\mathcal{E}} = \alpha_1 \sigma(Cms + Cps_1)$, it follows,

$$t_est - t_\tilde{act}_i \geq \frac{Cps}{Cps_i}\alpha_1 \sigma(Cms + Cps_1) - (\sum_{j=1}^{i} \alpha_j \sigma Cms + \alpha_i \sigma Cps)$$

Let $A = (\frac{Cps}{Cps_i}\alpha_1 - \alpha_1)\sigma Cms$ and let $B = \frac{Cps}{Cps_i}\alpha_1 \sigma Cps_1 - (\sum_{j=2}^{i} \alpha_i \sigma Cms + \alpha_i \sigma Cps)$. After simple algebraic manipulation, we have, $t_est - t_\tilde{act}_i \geq A + B$. So, to prove the Theorem, i.e. $t_est \geq t_act_i$, it is sufficient to prove $A \geq 0$ and $B \geq 0$. $\frac{Cps}{Cps_i} \geq 1$ directly leads to $A \geq 0$. From Eq. (3), it follows,

$$\alpha_1 \sigma(Cms + Cps_1) = \sum_{j=1}^{i} \alpha_j \sigma Cms + \alpha_i \sigma Cps_i.$$

That is,

$$\alpha_1 \sigma Cps_1 = \sum_{j=2}^{i} \alpha_j \sigma Cms + \alpha_i \sigma Cps_i.$$

Thus,

$$\begin{aligned} \frac{Cps}{Cps_i}\alpha_1 \sigma Cps_1 &= \frac{Cps}{Cps_i}(\sum_{j=2}^{i} \alpha_j \sigma Cms + \alpha_i \sigma Cps_i) \\ &= \frac{Cps}{Cps_i}\sum_{j=2}^{i} \alpha_j \sigma Cms + \alpha_i \sigma Cps \\ &\geq \sum_{j=2}^{i} \alpha_j \sigma Cms + \alpha_i \sigma Cps. \end{aligned}$$

It follows that,

$$\frac{Cps}{Cps_i}\alpha_1 \sigma Cps_1 - (\sum_{j=2}^{i} \alpha_i \sigma Cms + \alpha_i \sigma Cps) \geq 0$$

That is, $B \geq 0$. With $A \geq 0$ and $B \geq 0$, we have $t_est \geq t_\tilde{act}_i$. Since $t_\tilde{act}_i \geq t_act_i$, we conclude that $t_est \geq t_act_i$.

4.1.2 User-Split Partitioning

In this section, we present a common task partitioning method adopted by users of cluster-based research computing facilities like the U.S. CMS Tier-2 sites. Currently, a large CMS task is manually split by a user and the subtasks are then submitted to a cluster.

Based on the current practice, a User-Split algorithm is proposed. To emulate user behavior, the algorithm partitions a task into n equal-sized subtasks, where n is a user-specified number for requested nodes. The assumption is that a user will request n nodes that he or she thinks might be enough to satisfy the task deadline. It will fall in $[N_{min}, N]$ range, where N_{min} is the minimum number of nodes the task needs to meet its deadline if it starts execution immediately upon its arrival and N is the size of the cluster. Next, we analyze the algorithm for task completion time and N_{min} derivation.

Task Completion Time Analysis. Assume task $T = (A, \sigma, D)$ is split into n subtasks and each of them is assigned to a node. Then, the time node P_i completes its computation is,

$$C_i(\sigma, n) = s_i + \frac{\sigma Cms}{n} + \frac{\sigma Cps}{n}, \quad \text{for } i = 1, 2 \ldots, n$$

where s_i is the task start time for node P_i and the other two terms represent the transmission and computation times. Note that s_i may not be equal to r_i, the available time of node P_i, because the start time, s_i for data transmission to node P_i may be delayed by the transmission of data to node $P_1, P_2 \ldots, P_{i-1}$. Thus, we have $s_1 = r_1$ and $s_i = \max(r_i, s_{i-1} + \frac{\sigma Cms}{i})$, for $i = 2, \ldots, n$. And the completion time $C(\sigma, n)$ for task T is the maximum of $C_i(\sigma, n)$ for $i = 1, 2 \ldots, n$.

$$\text{That is,} \qquad C(\sigma, n) = s_n + \frac{\sigma Cms}{n} + \frac{\sigma Cps}{n} \tag{15}$$

Derivation of N_{min}. N_{min} denotes the minimum number of nodes task $T = (A, \sigma, D)$ needs to meet its deadline. The following constraint should be satisfied,

$$\sigma Cms + \frac{\sigma Cps}{N_{min}} \leq D$$

Thus, $\frac{\sigma Cps}{D - \sigma Cms} \leq N_{min}$. That is, $N_{min} = \lceil \frac{\sigma Cps}{D - \sigma Cms} \rceil$.

4.2 Algorithm Framework

As is typical for dynamic real-time scheduling algorithms [15, 23, 26], when a task arrives, the scheduler dynamically determines if it is feasible to schedule the new task without compromising the guarantees for previously admitted tasks. In [22] we have designed a general framework for a schedulability test, which can be configured to support various real-time divisible load scheduling algorithms by providing design decisions on: 1) scheduling policy (EDF or FIFO), 2) task partitioning rule (DLT-Based or User-Split partitioning), and 3) node assignment method (assigning a task $\tilde{n}^{min}$ or user-specified n nodes).

In this paper, we configure this framework (Figure 2) to generate two sets of algorithms that utilize IITs in a cluster. The first set of algorithms uses an EDF and the second set adopts a FIFO scheduling policy, where the task execution order is determined by task absolute deadlines or task arrival times. In addition, we have two different scheduling algorithms in each set: they are either 1) DLT-Based or 2)

Data Structure:

- AN(t) — the available number of idle processing nodes at time t.
- Release($node_k$) — the time the kth available node is released by a previous task.

Pseudocode:

```
boolean Schedulability-Test(NewTask)
    TempTaskList ← NewTask + TaskWaitingQueue

    // EDF or FIFO scheduling policy (Decision #1)
    order tasks in TempTaskList by their absolute deadlines or arrival times

    while TempTaskList != ϕ
        remove T_i(A_i, σ_i, D_i) from TempTaskList

        //Assign the task ñ_i^min or a user-specified number of nodes (Decision #3)
        n ← ñ_i^min (t) or a random number from [N_min, N] range

        identify the earliest time t when the available nodes AN(t) ≥ n

        // Set processor available times
        for k=1 to n
            r_k ← max(Release(node_k), A_i)
        end for

        // According to the chosen partition rule DLT-Based or User-Split partitioning
        (Decision #2), set expected completion time following Eq. (6) or Eq. (15)
        e_i ← Ê(σ_i, n) + r_n or C(σ_i, n)

        if e_i > A_i + D_i
            return false        // Deadline misses

        put T_i(A_i, σ_i, D_i, r_1, r_2, ..., r_n, n, e_i) into TempSchedule
    end while

    /* All tasks in the cluster are schedulable */
    Accept TempSchedule

    return true

end Schedulability Test()
```

Figure 2: Schedulability Test for the Algorithms.

User-Split based. According to the partitioning method adopted, our schedulers estimate task completion times following analysis in Section 4.1.1 or Section 4.1.2. The number of nodes assigned to each task is $\tilde{n}^{min}$ for DLT-Based algorithm and n, a random number in the range $[N_{min}..N]$ for the User-Split algorithm. Upon completion of the schedulability test, if all tasks are schedulable a feasible schedule is developed and the new task is accepted, otherwise, it is rejected.

By following the aforementioned framework, we generate four algorithms: EDF-DLT, EDF-UserSplit, FIFO-DLT, FIFO-UserSplit. The nomenclature for the algorithms includes two parts. The first part denotes the adopted scheduling policy: EDF or FIFO, while the second part represents the choice of the partitioning rule: DLT-Based partitioning (Section 4.1.1) or User-Split method (Section 4.1.2).

5 Performance Evaluation

In this section, we evaluate the proposed real-time scheduling algorithms: EDF-DLT and FIFO-DLT. First, we compare these two algorithms with the corresponding approaches: EDF-OPR-MN and FIFO-OPR-MN that we proposed in [22], which do not utilize IITs. EDF-OPR-MN was shown to be one of the best performing algorithms in [22]. However, it does not deal with the Inserted Idle Times problem. There are also other algorithms such as: EDF-OPR-AN and FIFO-OPR-AN that always execute a task on all N nodes in a cluster. These algorithms do not have the IITs problem, yet these are rarely adopted in real-life clusters due to obvious drawbacks and administration concerns. Second, we compare new algorithms against algorithms utilizing IITs; the User-Split algorithms: EDF-UserSplit and FIFO-UserSplit that were analyzed in Section 4.1.2.

Cluster Configuration. We use a discrete simulator to simulate a range of clusters that are compliant with the system model presented in Section 3. For every simulation, three parameters, N, C_{ms} and C_{ps} are specified for a cluster.

Workload Generation. To generate a set of tasks $T_i = (A_i, \sigma_i, D_i)$, we assume that the interarrival times follow an exponential distribution with a mean of $1/\lambda$, and task data sizes σ_i are assumed to be normally distributed with a specified mean of $Avg\sigma$ and a standard deviation equal to the mean. Task relative deadlines are assumed to be uniformly distributed in the range $[\frac{AvgD}{2}, \frac{3AvgD}{2}]$, where $AvgD$ is the mean relative deadline. To specify $AvgD$, we use the term $DCRatio$ [22]. It is defined as the ratio of mean deadline to mean minimum execution time (cost), that is $\frac{AvgD}{\mathcal{E}(Avg\sigma,N)}$, where $\mathcal{E}(Avg\sigma, N)$ is the execution time assuming the task has an average data size $Avg\sigma$ and is allocated to run on all N nodes simultaneously. Given a $DCRatio$, the cluster size N and the average data size $Avg\sigma$, $AvgD$ is implicitly specified as $DCRatio \times \mathcal{E}(Avg\sigma, N)$. This way, by $DCRatio$, task relative deadlines are specified relating to the average task execution time. In addition, a task relative deadline D_i is chosen to be larger than its minimum execution time $\mathcal{E}(\sigma_i, N)$. In summary, we could specify the following

parameters for a simulation: (*N*, C_{ms}, C_{ps}, $1/\lambda$, $Avg\sigma$, *DCRatio*).

To analyze the cluster load for a simulation, we use the metric $SystemLoad$ [22]. It is defined as, $SystemLoad = \frac{\mathcal{E}(Avg\sigma,N)}{\lambda}$, which is same as, $SystemLoad = \frac{TotalTaskNumber \times \mathcal{E}(N,Avg\sigma)}{TotalSimulationTime}$. For a simulation, we could specify $SystemLoad$ instead of average interarrival time $1/\lambda$. Configuring (*N*, C_{ms}, C_{ps}, *SystemLoad*, $Avg\sigma$, *DCRatio*) is equivalent to specifying (*N*, C_{ms}, C_{ps}, $1/\lambda$, $Avg\sigma$, *DCRatio*), because, $1/\lambda = \frac{SystemLoad}{\mathcal{E}(Avg\sigma,N)}$. To evaluate the performance of the real-time scheduling algorithms, we use the metric, *Task Reject Ratio*, defined as the ratio of the number of task rejections to the number of task arrivals. The smaller the *Task Reject Ratio*, the better the real-time scheduling algorithm.

For all figures in this paper, a point on a curve corresponds to the average performance of ten simulations.[2] In the ten runs, the same parameters (*N*, C_{ms}, C_{ps}, *SystemLoad*, $Avg\sigma$, *DCRatio*) are specified but different random numbers are generated for task arrival times A_i, data sizes σ_i, and deadlines D_i. For each simulation, the $TotalSimulationTime$ is 10,000,000 time units, which is sufficiently long.

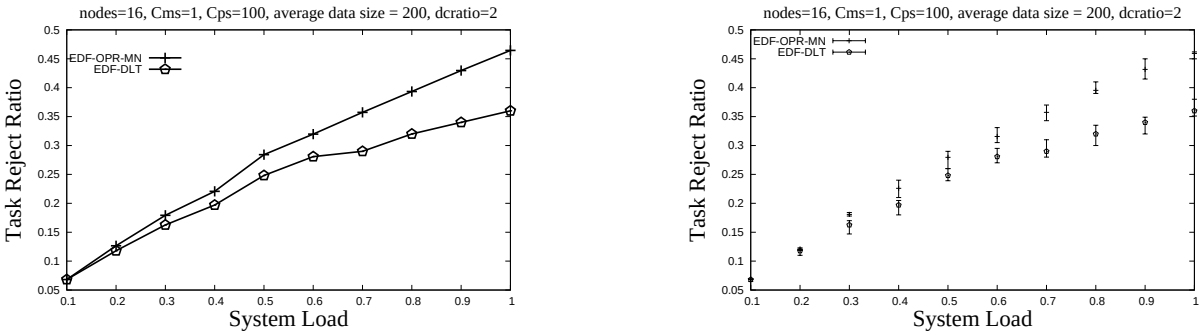
5.1 Benefits of Utilizing IITs

First we evaluate the performance of our new algorithms with respect to our previous approaches [22] where no IITs are utilized: EDF-DLT vs. EDF-OPR-MN and FIFO-DLT vs. FIFO-OPR-MN. In this section, we only report the comparison of EDF-DLT vs. EDF-OPR-MN here. The performance results for the other pair are similar and can be found in Appendix (Fig. 9 to 12).

For our **baseline model** we chose the following simulation parameters: number of processing nodes in the cluster, $N = 16$; unit data transmission time, $C_{ms} = 1$; unit data processing time, $C_{ps} = 100$; $SystemLoad$ changes in the range [0.1, 0.2, $\cdots$, 1.0]; Average data size, $Avg\sigma = 200$; and the ratio of the average deadline to the average execution time, $DCRatio = 2$. Our simulation has a two-fold objective. *First*, we want to verify our hypothesis that it is advantageous to utilize IITs in real-time cluster-based scheduling. *Second*, we study the effects of $DCRatio$.

To study the merits of utilizing IITs, we employ our baseline model. The two curves in Figure 3 show the *Task Reject Ratio* of algorithms: EDF-DLT and EDF-OPR-MN. Note that EDF-DLT always leads to a lower *Task Reject Ratio* than EDF-OPR-MN. Since EDF-OPR-MN has been one of the best performing algorithms proposed so far [22], our simulation result confirms our hypothesis that it is beneficial to utilize IITs in real-time cluster-based scheduling. By using IITs, the task execution time decreases and as a result the cluster can accommodate more tasks and meet their deadlines. We carried out the same type of simulations by changing, one at a time, the following cluster or workload parameters: *cluster size* N, *unit transmission time* Cms, *unit computation time* Cps and *average data size* $Avg\sigma$. Results are similar to Figure 3, showing EDF-DLT, the algorithm that utilizes IITs, always performs better (see Fig. 6 to 8 in Appendix for details).

[2] We report curves with 95% confidence intervals of baseline experiment in Fig. 3b.



(a) Baseline Experiment. (b) 95% Confidence Interval

Figure 3: Benefits of Utilizing IITs.

To study the effects of $DCRatio$ on our scheduling algorithms, we used the same configuration as the baseline model except that we varied the $DCRatio$ over [2,3,10,20,100] range. Results are presented in Figures 3, 4a-4d. We again observe that the EDF-DLT algorithm always performs better. In addition, we find that as the $DCRatio$ increases, the performance of EDF-DLT and EDF-OPR-MN converges. This is because the higher the $DCRatio$, the looser the task relative deadlines. Consequently, these two algorithms tend to allocate less nodes to a task. In general, the smaller the number of nodes assigned to a task the less the IITs. Thus, the benefits of utilizing IITs become less significant. In particular, when the $DCRatio$ is extremely high (equal to 100), the two algorithms perform almost the same (Figure 4d).

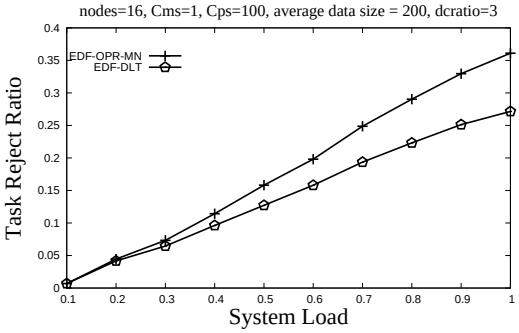
5.2 DLT-Based vs. User-Split Algorithms

This section evaluates the two partitioning methods: *DLT-Based and User-Split partitioning*. Both of these utilize IITs to compute arbitrarily divisible loads. The performance of the following algorithms is compared: EDF-DLT vs. EDF-UserSplit and FIFO-DLT vs. FIFO-UserSplit. Here we only show results for comparing EDF-based algorithms. Similar results were obtained for the other pair (see Fig. 13 to 16 in Appendix).

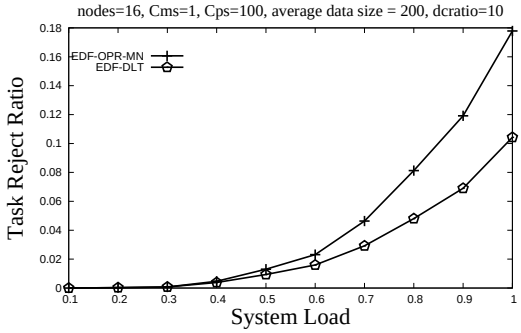
First, we conducted the simulation using the baseline model (Section 5.1). The two curves in Figure 5a show the *Task Reject Ratio* of the two algorithms. Observe that EDF-DLT always leads to smaller *Task Reject Ratios* than EDF-UserSplit, indicating our DLT-Based algorithm performs better.

The same type of simulations were carried out where we changed, one at a time, the following cluster or workload parameters: cluster size N, unit transmission time Cms, unit computation time Cps and average data size $Avg\sigma$. Results are similar to Figure 5a (refer to Fig. 13 to 16 in Appendix for details).

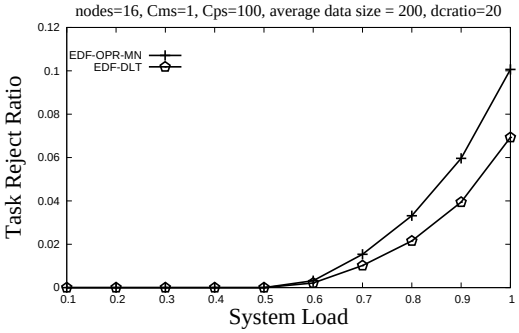
We also study the effects of changing $DCRatio$. When the $DCRatio$ is large, i.e. $DCRatio \geq 10$, sometimes the algorithm EDF-UserSplit performs better than EDF-DLT (Figure 5b). We conducted a total of 330 simulations with different system configurations. User-Split based algorithms perform better than the corresponding DLT-Based algorithms 8.22% of time. In addition when a DLT-Based algorithm



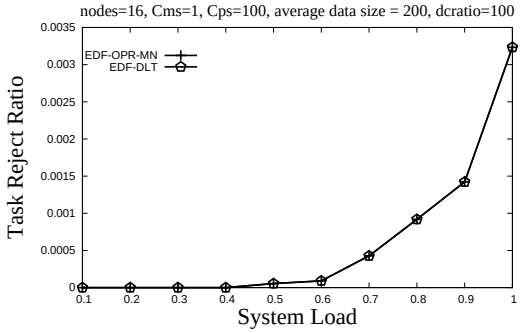
(a) Benefits of Utilizing IITs $DCRatio = 3$



(b) Benefits of Utilizing IITs $DCRatio = 10$



(c) Benefits of Utilizing IITs $DCRatio = 20$

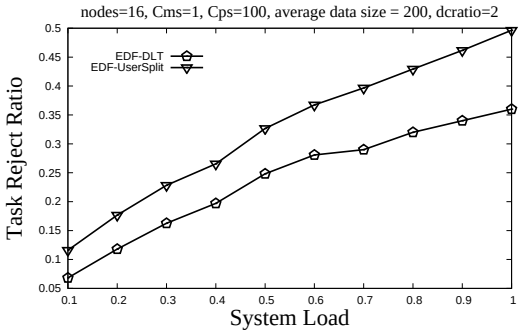


(d) Benefits of Utilizing IITs $DCRatio = 100$

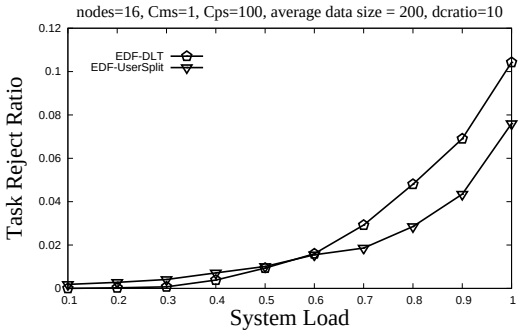
Figure 4: Benefits of Utilizing IITs: $DCRatio$ Effects.

performs better, its *Task Reject Ratio* is significantly lower than that of a User-Split algorithm. The average, maximum and minimum gains on *Task Reject Ratio* are 0.121, 0.224 and 0.003 respectively. On the other hand, when a User-Split algorithm performs better, only negligible average, maximum and minimum *Task Reject Ratio* gains are observed: 0.016, 0.028 and 0.003.

From the data, we can conclude that our DLT-Based approach has its advantages. Not only does it require no manual work by users as it automatically divides a task, but it also provides better performance most of the time. The reasons for its good performance are two-fold. First, our approach uses divisible load theory to guide task partitioning. Second, based on system load and a task deadline it dynamically



(a) Baseline Experiment



(b) $DCRatio = 10$

Figure 5: DLT-Based vs. User-Split Algorithms.

determines the number of nodes assigned to a task. This adaptive capability further makes our approach more appealing than the User-Split method.

6 Conclusion

In this paper, we address the classic Inserted Idle Times (IITs) problem in the context of real-time divisible load scheduling [22]. Two contributions are made. First, we propose a new approach to model the homogenous system with IITs as an equivalent heterogeneous system. Second, we prove that partitioning the task following the model and executing the subtasks in the homogenous cluster results in a task completion time earlier or equal to the estimate. This theorem in turn leads to a new real-time scheduling algorithm that utilizes the IITs. Intensive simulation results show that our approach does make use of the IITs to a large extent and significantly improves the system performance. We also compare our algorithm with the current practice of manually splitting a workload by the user. Simulation results demonstrate the advantage of our algorithm as compared to the user-split approach. Currently, we are working on expanding our approach to show, both theoretically and experimentally, that by adopting multi-round scheduling [10], we can further improve the IITs utilization and the system performance.

References

[1] T. Abdelzaher and V. Sharma. A synthetic utilization bound for aperiodic tasks with resource requirements. In *Proc. of 15th Euromicro Conference on Real-Time Systems*, pages 141–150, Portugal, July 2003.

[2] A. Amin, R. Ammar, and A. E. Dessouly. Scheduling real time parallel structure on cluster computing with possible processor failures. In *Proc of 9th IEEE Intl. Symp. on Computers and Comm.*, pages 62–67, 2004.

[3] Y. Amir, B. Awerbuch, A. Barak, R. Borgstrom, and A. Keren. An opportunity cost approach for job assignment in a scalable computing cluster. *IEEE Trans. on Parallel and Distributed Systems*, 11(7), 2000.

[4] R. A. Ammar and A. Alhamdan. Scheduling real time parallel structure on cluster computing. In *Proc. of 7th IEEE International Symposium on Computers and Communications*, pages 69–74, Italy, July 2002.

[5] J. Anderson and A. Srinivasan. Pfair scheduling: Beyond periodic task systems, 2000.

[6] ATLAS (AToroidal LHC Apparatus) Experiment, CERN (European Lab for Particle Physics).

[7] L. Barsanti and A. C. Sodan. Adaptive job scheduling strategies via predictive job resource allocation. In *Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), in conjunction with ACM SIGMETRICS*, Saint Malo, France, June 2006.

[8] V. Bharadwaj and G. Barlas. Scheduling divisible loads with processor release times and finite size buffer capacity constraints. *special issue of Cluster Computing on Divisible Load Scheduling, Kluwer Academic Publishers, Vol. 6, No. 1*, 2003.

[9] V. Bharadwaj, H. Li, and T. Radhakrishnan. Scheduling divisible loads in bus networks with arbitrary processor release times. *Computers and Math. with Applications, Pergamon Press, Vol. 32, No. 7*, 1996.

[10] V. Bharadwaj, T. G. Robertazzi, and D. Ghose. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press, Los Alamitos, CA, 1996.

[11] V. Bharadwaj, B. Veeravalli, and W. Min. Scheduling divisible loads on heterogeneous linear daisy chain networks with arbitrary processor release times. *IEEE Trans. on Parallel and Dist. Systs., vol. 15, no. 3*, 04.

[12] B. N. Chun and D. E. Culler. Market-based proportional resource sharing for clusters. Technical Report UCB/CSD-00-1092, EECS Department, University of California, Berkeley, 2000.

[13] W. Cirne. Using moldability to improve the performance of supercomputer jobs, 2001.

[14] 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/.

[15] M. L. Dertouzos and A. K. Mok. Multiprocessor online scheduling of hard-real-time tasks. *IEEE Trans. Softw. Eng.*, 15(12):1497–1506, 1989.

[16] M. Eltayeb, A. Dogan, and F. Özgüner. A data scheduling algorithm for autonomous distributed real-time applications in grid computing. In *Proc. of 33rd International Conference on Parallel Processing*, pages 388–395, Montreal, Canada, Aug 2004.

[17] S. Funk and S. Baruah. Task assignment on uniform heterogeneous multiprocessors. In *Proc of 17th Euromicro Conference on Real-Time Systems*, pages 219–226, July 2005.

[18] L. He, S. A. Jarvis, D. P. Spooner, X. Chen, and G. R. Nudd. Hybrid performance-oriented scheduling of moldable jobs with qos demands in multiclusters and grids. In *GCC*, pages 217–224, 2004.

[19] D. Isovic and G. Fohler. Efficient scheduling of sporadic, aperiodic, and periodic tasks with complex constraints. In *Proc. of 21st IEEE Real-Time Systems Symposium*, Orlando, FL, Nov 2000.

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

[21] D. Lifka. The anl/ibm sp scheduling system. *Job Scheduling Strategies for Parallel Processing, D. G. Feitelson and L. Rudolph (eds.), pp. 295C 303, Springer-Verlag, Lect. Notes Comput. Sci. vol. 949.*, 1995.

[22] X. Lin, Y. Lu, J. Deogun, and S. Goddard. Real-time divisible load scheduling for cluster computing. In *13th IEEE Real-Time and Embedded Technology and Application Symposium*, Bellevue, WA, April 2007.

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

[24] A. Mu'alem and D. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling, 2001.

[25] X. Qin and H. Jiang. Dynamic, reliability-driven scheduling of parallel real-time jobs in heterogeneous systems. In *Proc. of 30th International Conf. on Parallel Processing*, pages 113–122, Spain, Sep 2001.

[26] K. Ramamritham, J. A. Stankovic, and P. fei Shiah. Efficient scheduling algorithms for real-time multiprocessor systems. *IEEE Trans. on Parallel and Distributed Systems*, 1(2):184–194, Apr 1990.

[27] K. Ramamritham, J. A. Stankovic, and W. Zhao. Distributed scheduling of tasks with deadlines and resource requirements. *IEEE Trans. Comput.*, 38(8):1110–1123, 1989.

[28] T. G. Robertazzi. Ten reasons to use divisible load theory. *Computer*, 36(5):63–68, 2003.

[29] R. K. S. Srinivasan. Selective reservation strategies for backfill job scheduling. *Job Scheduling Strategies for Parallel Processing, D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn (eds.), pp. 55C 71, Springer-Verlag, 2002. Lect. Notes Comput. Sci. vol. 2537*, 2002.

[30] G. Sabin, M. Lang, and P. Sadayappan. Moldable parallel job scheduling using job efficiency: An iterative approach. In *Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), in conjunction with ACM SIGMETRICS*, Saint Malo, France, June 2006.

[31] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya. Libra: a computational economy-based job scheduling system for clusters. *Software: Practice and Experience*, 34(6):573–590, 2004.

[32] S. Srinivasan, S. Krishnamoorthy, and P. Sadayappan. A robust scheduling strategy for moldable scheduling of parallel jobs. In *CLUSTER*, pages 92–99, 2003.

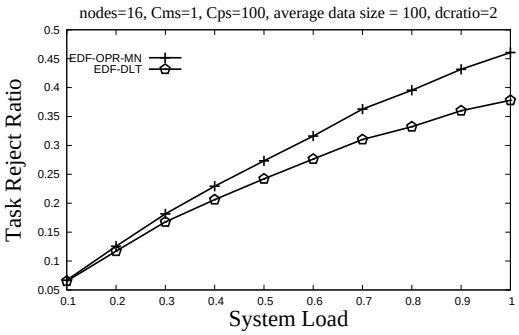
[33] D. Swanson. Personal communication. Director, UNL Research Computing Facility (RCF) and UNL CMS Tier-2 Site, Aug 2005.

[34] 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.

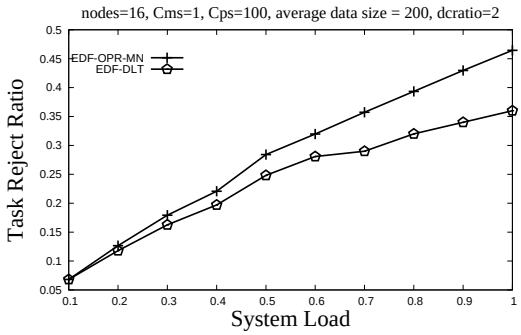
[35] C. S. Yeo and R. Buyya. A taxonomy of market-based resource management systems for utility-driven cluster computing. *Software: Practice and Experience*, accepted in Sep 2005.

[36] L. Zhang. Scheduling algorithm for real-time applications in grid environment. In *Proc. of IEEE International Conference on Systems, Man and Cybernetics*, Oct 2002.

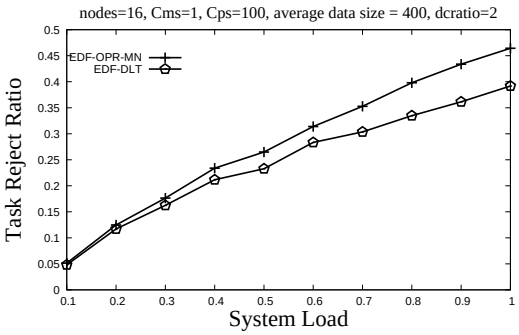
APPENDIX



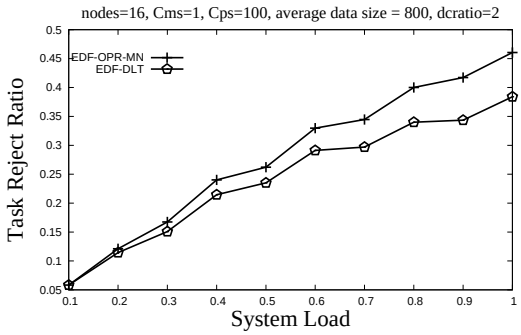
(a) Benefits of Utilizing IITs $Avg\sigma = 100$



(b) Benefits of Utilizing IITs $Avg\sigma = 200$

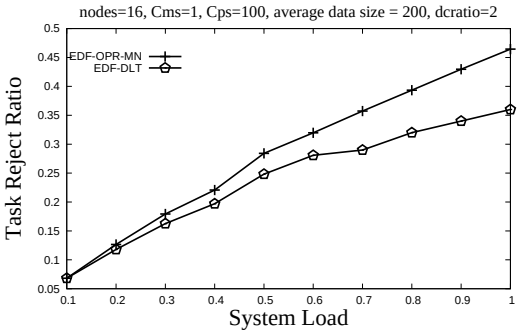


(c) Benefits of Utilizing IITs $Avg\sigma = 400$

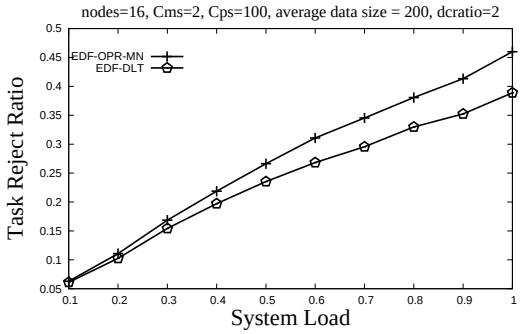


(d) Benefits of Utilizing IITs $Avg\sigma = 800$

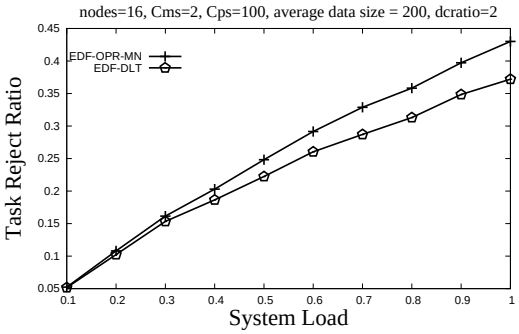
Figure 6: Benefits of Utilizing IITs: $Avg\sigma$ Effects.



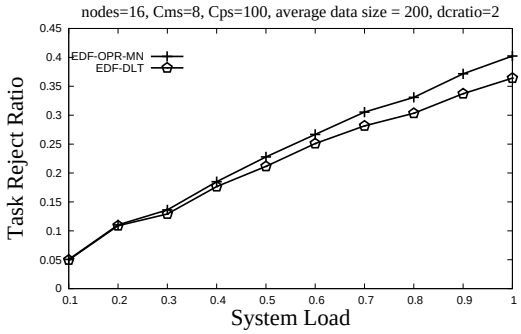
(a) Benefits of Utilizing IITs $Cms = 1$



(b) Benefits of Utilizing IITs $Cms = 2$

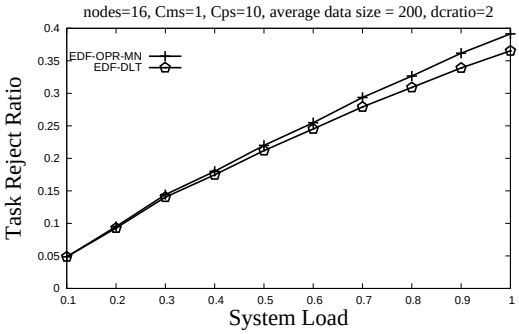


(c) Benefits of Utilizing IITs $Cms = 4$

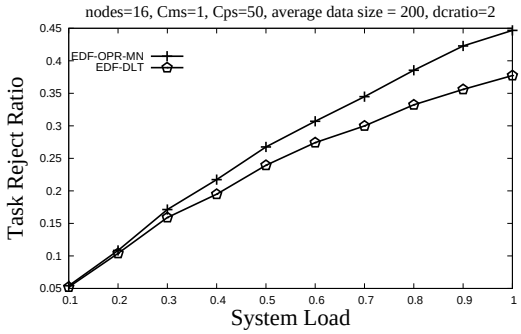


(d) Benefits of Utilizing IITs $Cms = 8$

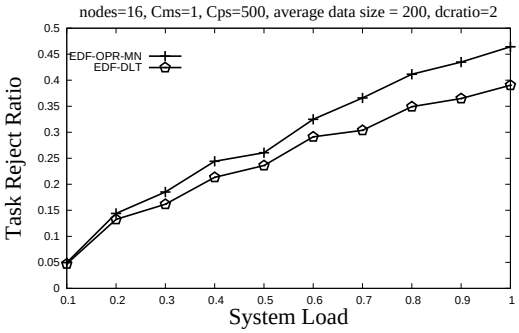
Figure 7: Benefits of Utilizing IITs: Cms Effects.



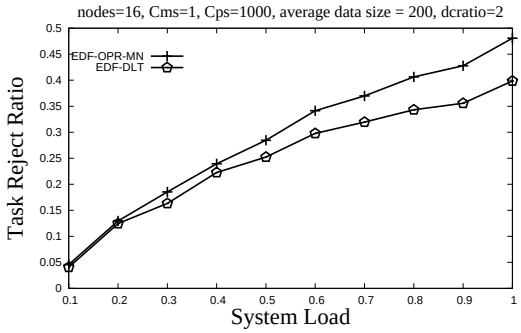
(a) Benefits of Utilizing IITs $Cps = 10$



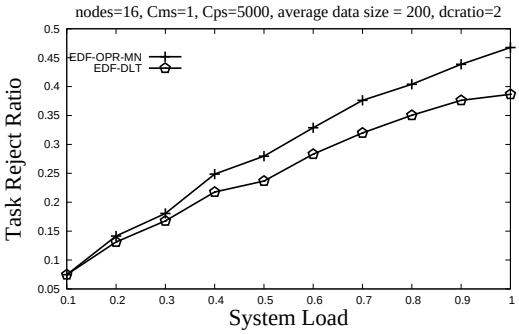
(b) Benefits of Utilizing IITs $Cps = 50$



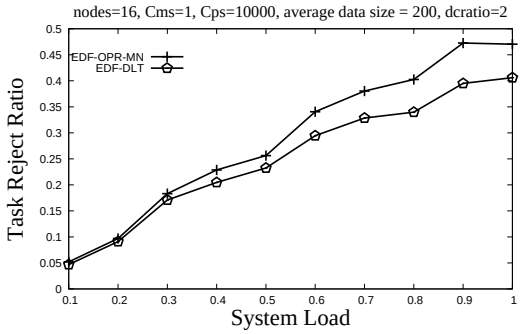
(c) Benefits of Utilizing IITs $Cps = 500$



(d) Benefits of Utilizing IITs $Cps = 1000$

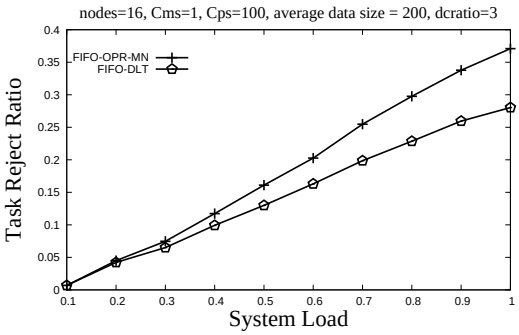


(e) Benefits of Utilizing IITs $Cps = 5000$

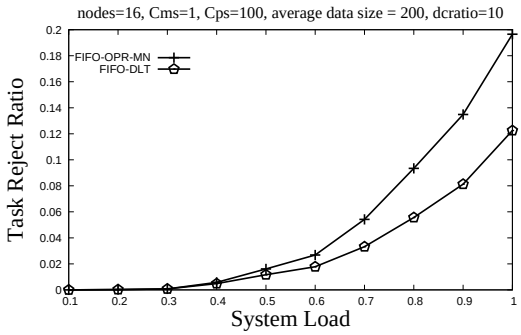


(f) Benefits of Utilizing IITs $Cps = 10000$

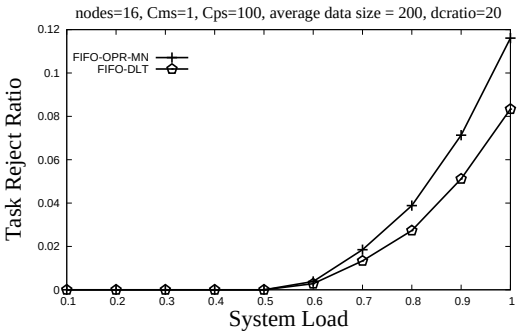
Figure 8: Benefits of Utilizing IITs: Cps Effects.



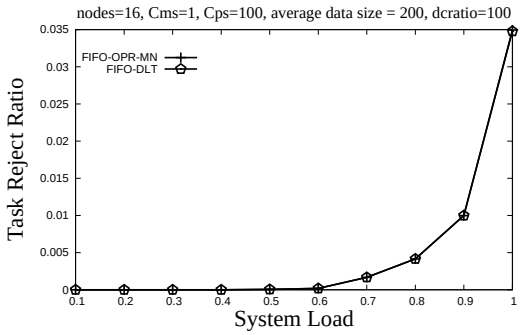
(a) Benefits of Utilizing IITs $DCRatio = 3$



(b) Benefits of Utilizing IITs $DCRatio = 10$

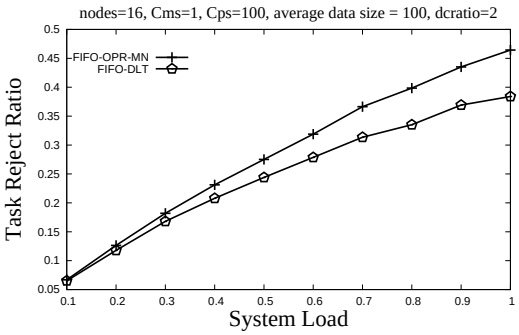


(c) Benefits of Utilizing IITs $DCRatio = 20$

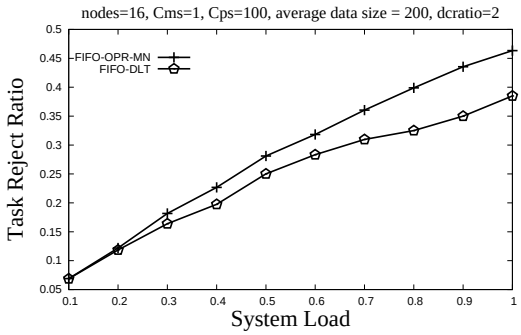


(d) Benefits of Utilizing IITs $DCRatio = 100$

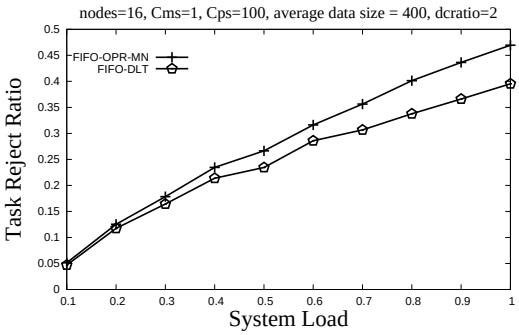
Figure 9: Benefits of Utilizing IITs: $DCRatio$ Effects.



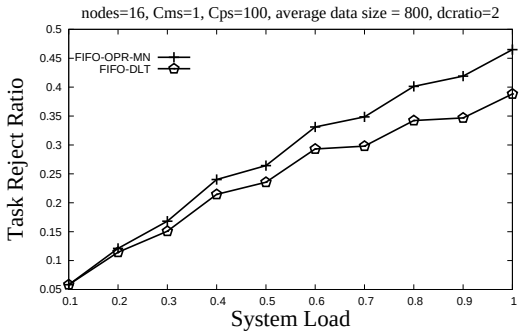
(a) Benefits of Utilizing IITs $Avg\sigma = 100$



(b) Benefits of Utilizing IITs $Avg\sigma = 200$

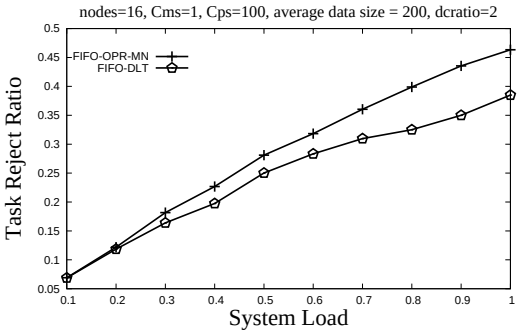


(c) Benefits of Utilizing IITs $Avg\sigma = 400$

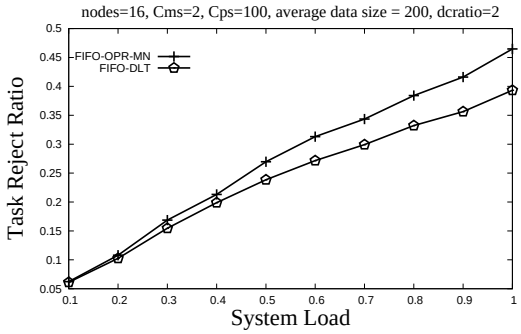


(d) Benefits of Utilizing IITs $Avg\sigma = 800$

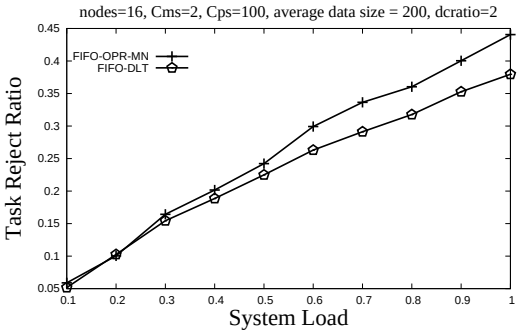
Figure 10: Benefits of Utilizing IITs: $Avg\sigma$ Effects.



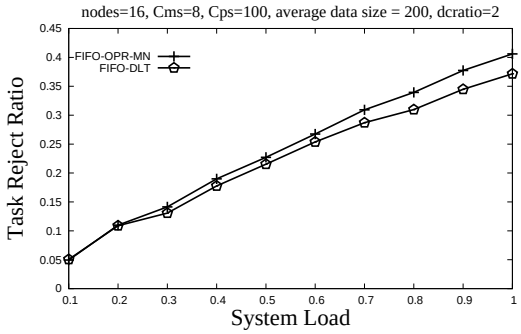
(a) Benefits of Utilizing IITs $Cms = 1$



(b) Benefits of Utilizing IITs $Cms = 2$

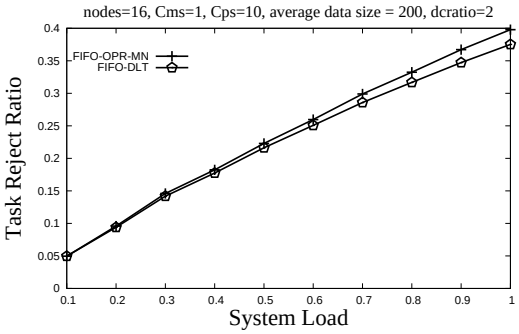


(c) Benefits of Utilizing IITs $Cms = 4$

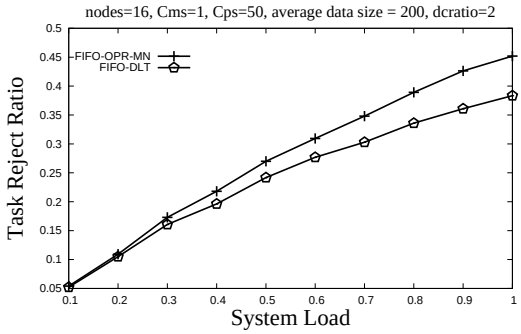


(d) Benefits of Utilizing IITs $Cms = 8$

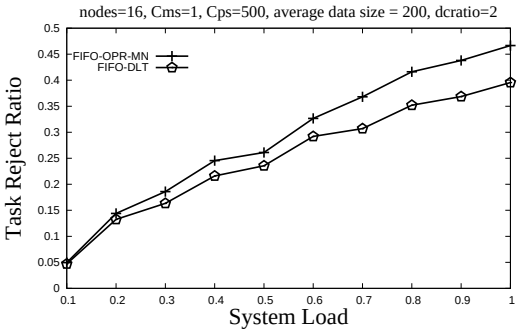
Figure 11: Benefits of Utilizing IITs: Cms Effects.



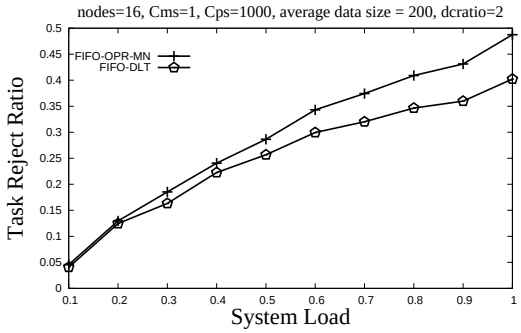
(a) Benefits of Utilizing IITs $Cps = 10$



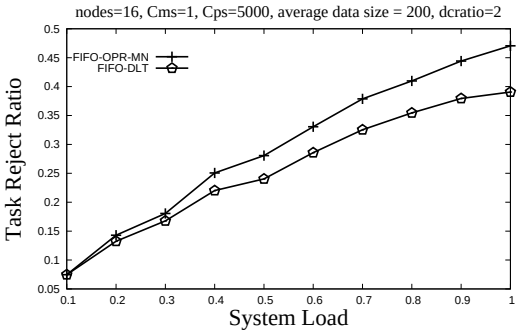
(b) Benefits of Utilizing IITs $Cps = 50$



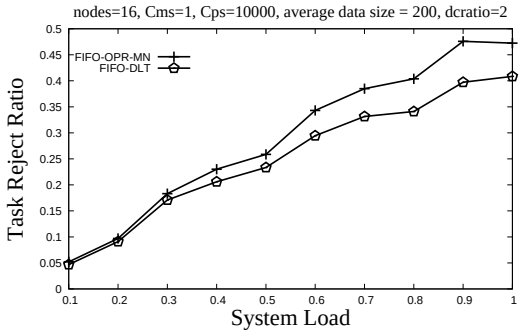
(c) Benefits of Utilizing IITs $Cps = 500$



(d) Benefits of Utilizing IITs $Cps = 1000$

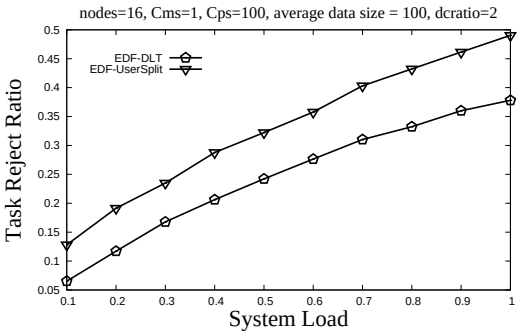


(e) Benefits of Utilizing IITs $Cps = 5000$

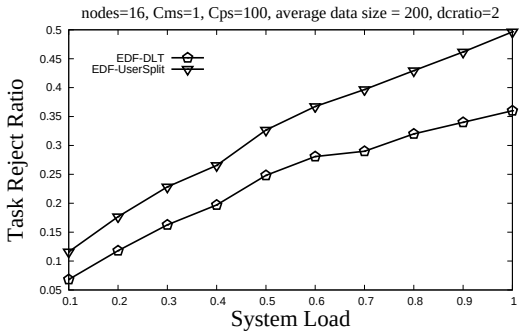


(f) Benefits of Utilizing IITs $Cps = 10000$

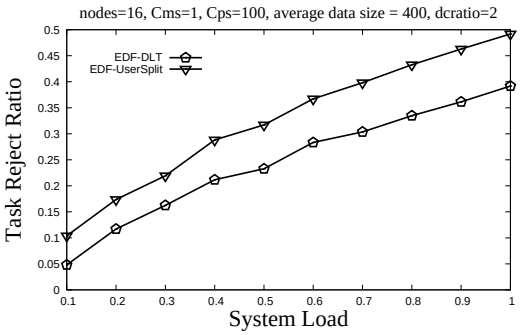
Figure 12: Benefits of Utilizing IITs: Cps Effects.



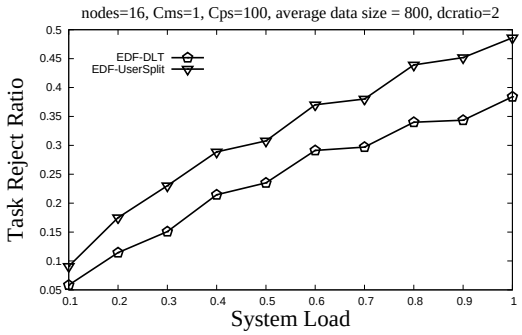
(a) DLT-Based vs. User-Split, $Avg\sigma = 100$



(b) DLT-Based vs. User-Split, $Avg\sigma = 200$

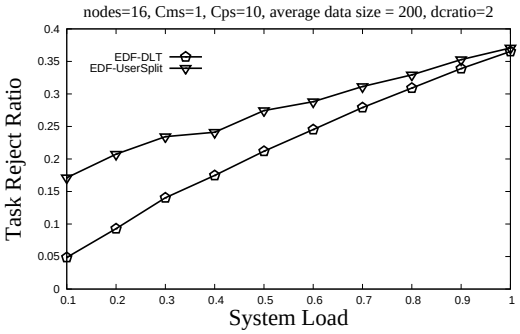


(c) DLT-Based vs. User-Split, $Avg\sigma = 400$

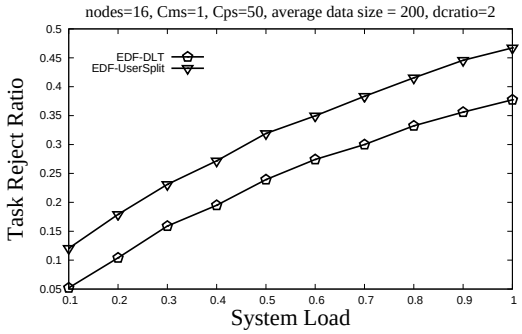


(d) DLT-Based vs. User-Split, $Avg\sigma = 800$

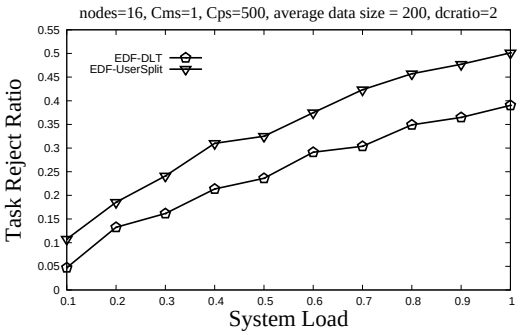
Figure 13: DLT-Based vs. User-Split: $Avg\sigma$ Effects



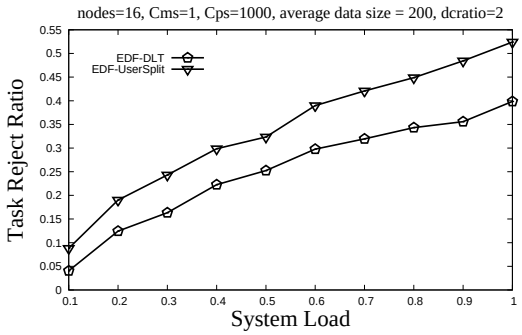
(a) DLT-Based vs. User-Split, $Cps = 10$



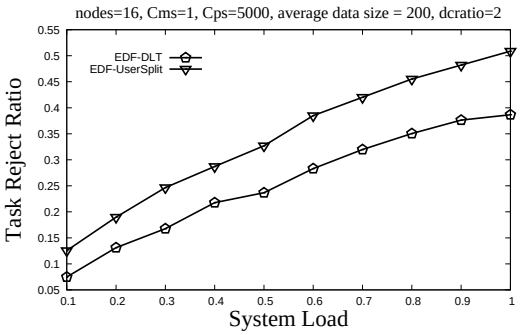
(b) DLT-Based vs. User-Split, $Cps = 50$



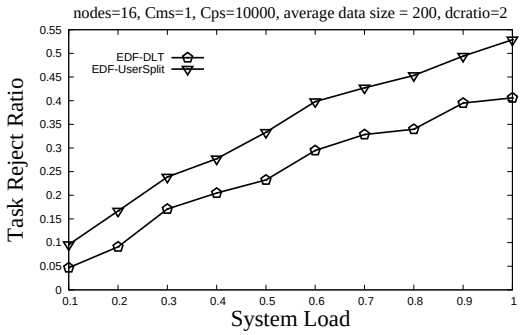
(c) DLT-Based vs. User-Split, $Cps = 500$



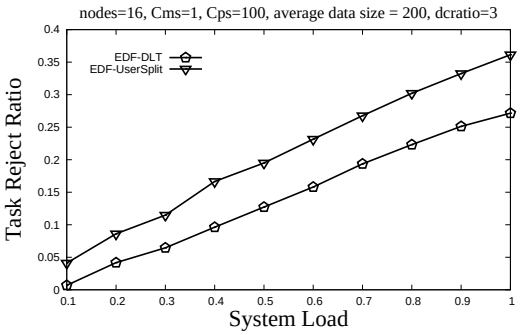
(d) DLT-Based vs. User-Split, $Cps = 1000$



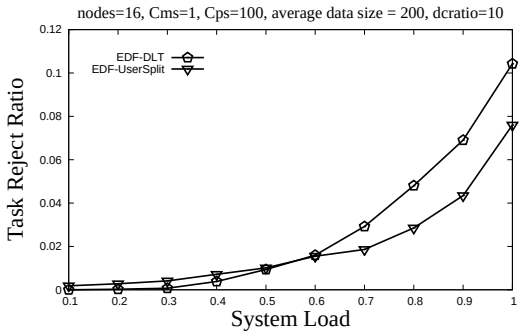
(e) DLT-Based vs. User-Split, $Cps = 5000$



(f) DLT-Based vs. User-Split, $Cps = 10000$

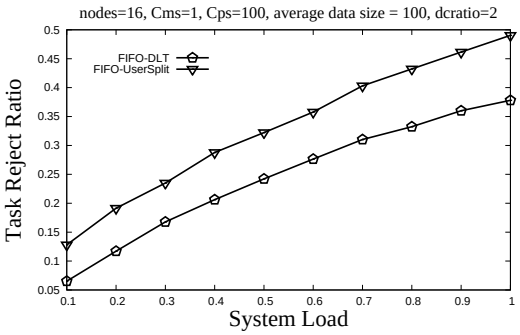


(g) DLT-Based vs. User-Split, $DCRatio = 3$

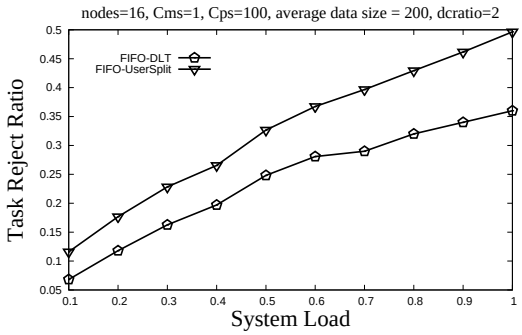


(h) DLT-Based vs. User-Split, $DCRatio = 10$

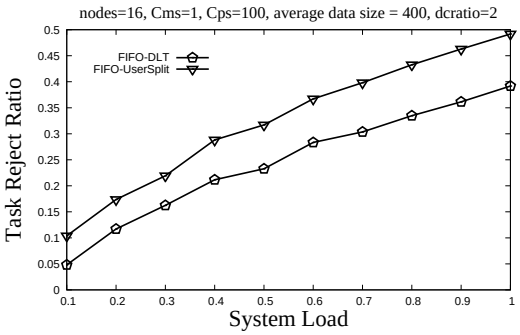
Figure 14: DLT-Based vs. User-Split Algorithms



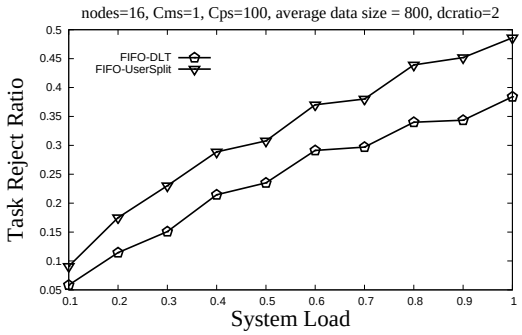
(a) DLT-Based vs. User-Split, $Avg\sigma = 100$



(b) DLT-Based vs. User-Split, $Avg\sigma = 200$

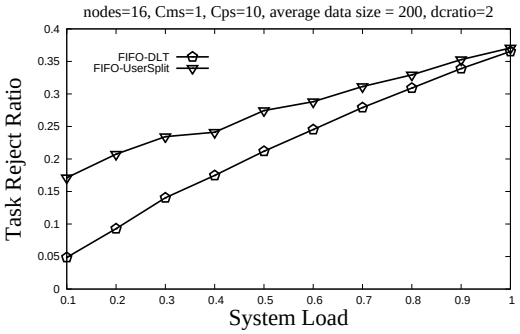


(c) DLT-Based vs. User-Split, $Avg\sigma = 400$

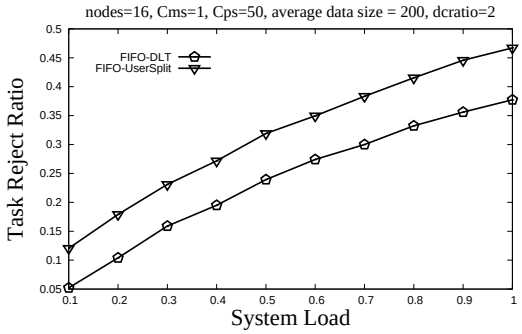


(d) DLT-Based vs. User-Split, $Avg\sigma = 800$

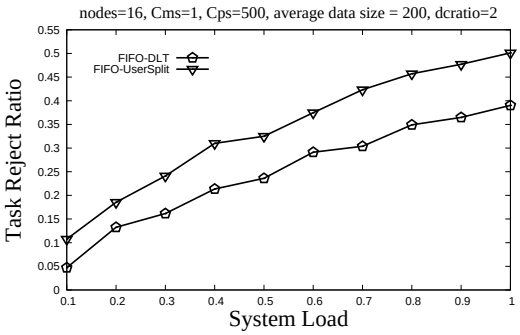
Figure 15: DLT-Based vs. User-Split: $Avg\sigma$ Effects



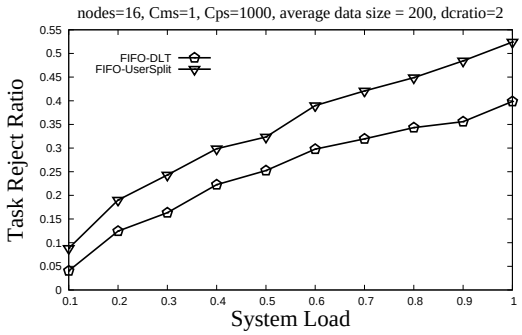
(a) DLT-Based vs. User-Split, $Cps = 10$



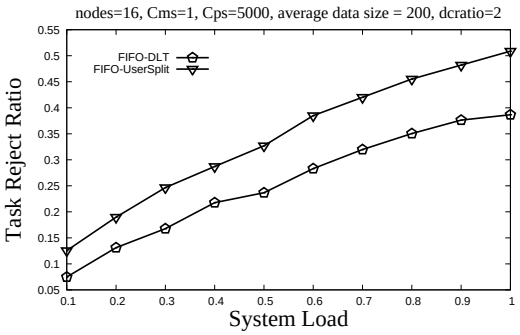
(b) DLT-Based vs. User-Split, $Cps = 50$



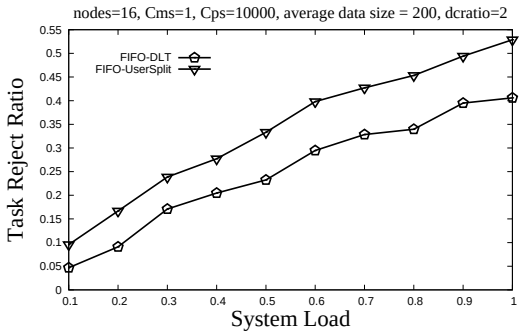
(c) DLT-Based vs. User-Split, $Cps = 500$



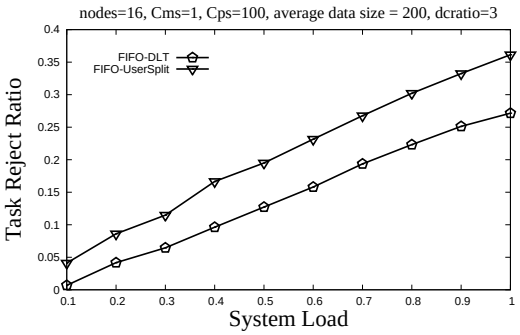
(d) DLT-Based vs. User-Split, $Cps = 1000$



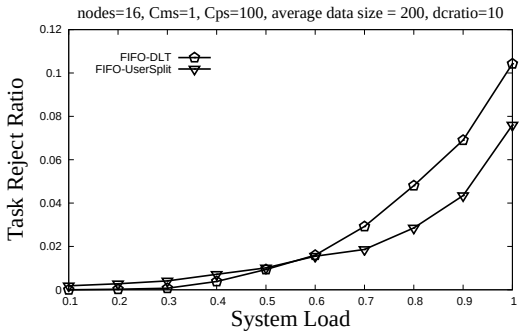
(e) DLT-Based vs. User-Split, $Cps = 5000$



(f) DLT-Based vs. User-Split, $Cps = 10000$



(g) DLT-Based vs. User-Split, $DCRatio = 3$



(h) DLT-Based vs. User-Split, $DCRatio = 10$

Figure 16: DLT-Based vs. User-Split Algorithms