

# Enhancing Real-time Scheduling of Divisible Loads by Utilizing Inserted Idle Time

Xuan Lin, Ying Lu, Jitender Deogun, Steve Goddard  
*{lxuan, ylu, deogun, goddard}@cse.unl.edu*

Department of Computer Science and Engineering  
University of Nebraska - Lincoln, Lincoln, NE 68588

**Abstract.** Providing QoS and performance guarantees for arbitrarily divisible loads in a cluster has become a significant problem. While progress is being made in scheduling arbitrarily divisible loads, some of the proposed approaches may cause Inserted Idle Times (IITs) that are detrimental to system performance. Two contributions are made in addressing this problem. First, we propose two constraints that, when satisfied, lead to an optimal partitioning in utilizing IITs. Second, we integrate the new partitioning method with a previous approach and develop an enhanced algorithm that better utilizes IITs. Simulation results demonstrate the advantages of our new approach.

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

## 1 Introduction

Arbitrarily divisible applications consist of an amount of data that can be divided arbitrarily into a desirable number of independent load fractions, and each sub-task (fraction) itself is arbitrarily divisible. This perfectly parallel model is a good representation of many scientific applications that consist of huge numbers of identical, low-granularity loads. For example, the CMS (Compact Muon Solenoid) [1] and ATLAS (AToroidal LHC Apparatus) [2] projects, associated with the LHC (Large Hadron Collider) at CERN (European Laboratory for Particle Physics), execute cluster-based applications with arbitrarily divisible loads. Usually, such applications require a large amount of resources and can only be deployed in commodity clusters or computational grids.

To efficiently utilize large-scale clusters, an on-line resource management system (RMS) is needed to provide real-time guarantees or QoS. This is becoming a significant issue for research computing facilities, such as the U.S. CMS Tier-2 sites [3], that execute large numbers of arbitrarily divisible loads. Thus, researchers, e.g., [4, 5], have begun to investigate real-time divisible load scheduling, with significant initial progress in important theories and applications.

However, the challenge, to efficiently schedule a job when there are not enough resources, has not yet been adequately addressed for real-time divisible loads. When scheduling a parallel job, if a sufficient number of processors are

available, the processors are allocated and the job is started. But if the required number of processors is not available, the job waits for additional processors. This essentially leads to a waste of processing power as some processors sit idle waiting to start the job. This is a system inefficiency that we refer to as the Inserted Idle Times (IITs) problem [6]. To alleviate this limitation, backfilling algorithms [7] 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 have previously developed a real-time scheduling algorithm that utilizes IITs [6]. Although the approach has significantly improved the system performance, it cannot fully utilize IITs. In this paper, we propose a new strategy to further make use of IITs. Not only can our enhanced algorithm schedule real-time divisible loads with different processor available times, when certain conditions hold, it can also optimally partition and schedule jobs to fully utilize IITs. Two contributions are made in this paper. First, we propose a new partitioning approach to fully utilize IITs and investigate its applicability constraints. Second, we integrate this with our previous work [6] and propose a new real-time scheduling algorithm.

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. Section 4 discusses real-time scheduling algorithms investigated in this paper. We evaluate the algorithms performance in Section 5 and conclude the paper in Section 6.

## 2 Related Work

The scheduling models investigated for real-time distributed systems most often (e.g., [8]) assume periodic or aperiodic sequential jobs where each job is allocated to a single resource and must be executed by its deadline. With the evolution of cluster computing, researchers have begun to investigate real-time scheduling of parallel applications. However, each 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 [4], to the best of our knowledge, only a few researchers [5, 9] have investigated the real-time scheduling of arbitrarily divisible loads.

Utility-driven cluster computing has been well researched [10] to improve the utility delivered to users. Proposed cluster RMSs [11] 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 most closely related work to ours is the scheduling of “scalable tasks” [9] or “moldable jobs” [12], where only a few papers [9] have considered QoS support. In [5] we investigated real-time cluster-based divisible load scheduling and proposed several algorithms for homogenous clusters. In [6], we developed a real-time scheduling approach that utilizes Inserted Idle Times (IITs). A mechanism to utilize processor idle-times, also called fragments, was investigated in [9], wherein a task is assigned a larger number of nodes to utilize more process-

ing power. Complementary to that approach, our algorithm in [6] enables a task to utilize a processor as soon as it becomes available. While results in [9] show that the performance improvement of their approach is negligible, our previous approach [6] has led to much better performance, even though it cannot fully utilize IITs. In this paper, we propose a new partitioning approach. We prove that if certain constraints hold, any task can be partitioned optimally to fully make use of IITs. By integrating such an optimal partitioning method with algorithms we have proposed in [6], system resources are better utilized and the performance is further improved.

Divisible load theory (DLT) provides an in-depth study of distribution strategies for arbitrarily divisible loads [13]. In our previous work [5], 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 our new partitioning algorithm.

### 3 Task and System Models

In this paper, we adopt the same task and system models as our previous work [5, 6]. For completeness, we briefly present these below.

**Task Model.** Similar to the classic real-time aperiodic task model, we assume each aperiodic task  $T_i$  consists of a single invocation specified by the tuple  $(A_i, \sigma_i, D_i)$ , where  $A_i$  is the task arrival time,  $\sigma_i$  is the total data size of the task, and  $D_i$  is its relative deadline. The task absolute deadline is given by  $A_i + D_i$ . We present in Section 4 how task execution time is dynamically computed based on total data size  $\sigma_i$ , resources allocated (i.e., processing nodes and bandwidth), and the partitioning method applied to parallelize the computation.

**System Model.** We consider a common cluster model, which consists of a head node, denoted by  $P_0$ , connected via a switch to  $N$  processing nodes, denoted by  $P_1, P_2, \dots, P_N$ . We assume a homogeneous model in which all processing nodes have the same computational power and all links from the switch to the processing nodes have the same bandwidth. Like a typical cluster environment, the system model assumes 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.

As with divisible load theory, we use linear models to represent processing and transmission times [14]. In the simplest scenario, the computation time of a load  $\sigma$  is calculated by a cost function  $Cp(\sigma) = \sigma\chi$ , where  $\chi$  represents the time to compute a unit of workload on a single processing node. The communication time of a load  $\sigma$  is calculated by a cost function  $Cm(\sigma) = \sigma\tau$ , where  $\tau$  is the time to transmit a unit of workload from the head node to a processing node.

The following notations, partially adopted from [14], are used in this paper.

- $T = (A, \sigma, D)$ : A divisible task, where  $A$  is the arrival time,  $\sigma$  is the data size, and  $D$  is the relative deadline;
- $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_n)$ : Data distribution vector, where  $n$  is the number of processing nodes allocated to the task,  $\alpha_j$  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,  $0 < \alpha_j \leq 1$  and  $\sum_{j=1}^n \alpha_j = 1$ ;
- $\tau$ : Cost of transmitting a unit workload;
- $\chi$ : 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. Due to space limitations, we omit the proofs of the theorems in this Section.

### 4.1 Real-Time Divisible Load Scheduling

An admission controller, which is a part of our scheduler, executes on the head node. As is typical for dynamic real-time scheduling algorithms [15–17], 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. The task’s schedulability test will be described in Section 4.5.

In [5], 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, such as 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 [5] we have shown that assigning only the minimum number of nodes to a task to meet its deadline is an efficient approach. However, this strategy, as with many others, leads to the IITs problem. To utilize IITs, we focused on the second module in [6]. That is, we designed a new task partitioning module for real-time divisible load scheduling. However, that approach has limitations. First, it does not guarantee optimal partitioning [14] and cannot ensure all assigned nodes complete their computations at the same time. Second, not all IITs are fully utilized. In the following sections, we first find the scenario when the IITs can be fully utilized and propose an optimal partitioning approach to fully make use of IITs. We then integrate this partitioning method with algorithms proposed in [6] to improve the system’s real-time performance.

### 4.2 Inserted Idle Times Problem

The IITs problem occurs when the number of processors available is less than that required by the next job. Many parallel job scheduling algorithms [7, 5] lead to this problem. If no strategy is implemented to make use of IITs, the job has to wait until enough processors become available, which leads to a waste

of processing power as some processors are idle waiting. Backfilling [7] is an approach proposed in the literature to alleviate this problem. It is a general approach applicable to all types of parallel jobs — whether task graph based and modularly divisible or arbitrarily divisible.

An arbitrarily divisible load, however, has a unique property, that is, it can be arbitrarily partitioned into a large number of independent subtasks of arbitrarily small size. Thus, the subtasks can be scheduled flexibly and independently. Exploiting this property of arbitrarily divisible loads, we have proposed algorithms [6] that schedule divisible loads with different processor available times and utilize IITs in a cluster.

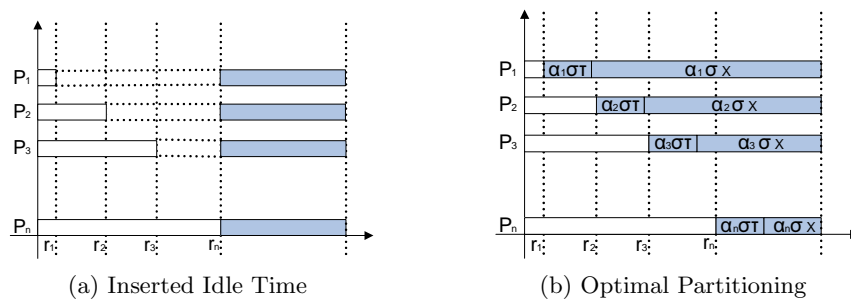


Fig. 1: Timing Diagrams

As shown in Figure 1a, when a task arrives at time  $r_1$ , the scheduler decides to assign the task  $n$  processors. However, it is not until time  $r_n$  that all of the  $n$  processors become available. We denote the available time of the  $i^{th}$  processor as  $r_i$ ,  $i = 1, \dots, n$ . Without loss of generality, we assume the processors are ordered by their available times, that is

$$r_1 \leq r_2 \leq \dots r_i \leq \dots \leq r_n$$

When scheduling the task, if we do not make use of the processors before time  $r_n$ , as shown in Figure 1a, the task will not start until time  $r_n$ . Let  $p_i = r_n - r_i$ ,  $i = 1, 2, \dots, n$ . Since the  $i^{th}$  processor is idle during the time period  $[r_i, r_n]$ , the total wasted cycles for these  $n$  processors is  $\sum_{i=1}^n p_i$ , which leads to sub-optimal cluster performance.

### 4.3 Optimal Partitioning

The unique property of a divisible load provides flexibility in scheduling. We can start part of the load on a processor as soon as the processor becomes available. However, the challenge is to optimally partition the load among the  $n$  assigned processors. According to DLT, the optimal execution time is obtained when all nodes allocated to a divisible task finish their computations at the same time

[14]. Let  $s_i$  represent the start time of the  $i^{th}$  node and  $\mathcal{E}(\sigma, n)$  denote the task completion time. To ensure all nodes finish at the same time, we have

$$\begin{aligned}\mathcal{E}(\sigma, n) &= s_1 + \alpha_1\sigma(\tau + \chi) = s_2 + \alpha_2\sigma(\tau + \chi) = \dots \\ &= s_n + \alpha_n\sigma(\tau + \chi).\end{aligned}\tag{1}$$

If any processor  $i$  can start as soon as it becomes available at time  $r_i$  (i.e.,  $s_i = r_i$ ,  $i=1, \dots, n$ ), we can fully utilize IITs. Following DLT, with no IITs, an optimal partitioning will result in the task completion time of Equation (2).

$$\begin{aligned}\mathcal{E}(\sigma, n) &= r_1 + \alpha_1\sigma(\tau + \chi) = r_2 + \alpha_2\sigma(\tau + \chi) = \dots \\ &= r_n + \alpha_n\sigma(\tau + \chi)\end{aligned}\tag{2}$$

**Theorem 1.** *If  $\forall i : 1 \leq i \leq n$ ,  $s_i = r_i$ , Equations (3) and (4) result in an optimal partitioning of the load with no unused IITs.*

$$\alpha_1 = \frac{1}{n} + \frac{(n-1)p_1 - \sum_{i=2}^n p_i}{n\sigma(\tau + \chi)}\tag{3}$$

$$\alpha_i = \alpha_1 - \frac{p_1 - p_i}{\sigma(\tau + \chi)} \quad \text{where } p_i = r_n - r_i.\tag{4}$$

Figure 1b shows the task execution time diagram following this optimal partitioning scheme. We can see in this scenario every processor is busy for either its data transmission or its subtask computation.

#### 4.4 Constraints for Optimal Partitioning

The start time may not always be equal to the release time, as in our model data transmission is not parallel, the start time  $s_i$  of the  $i^{th}$  node may be delayed by data transmissions to the  $1^{st}, 2^{nd}, \dots, (i-1)^{th}$  nodes. Thus, we have

$$s_1 = r_1, s_i = \max(r_i, s_{i-1} + \alpha_{i-1}\sigma\tau), i = 2, \dots, n$$

And in the worst case when  $r_1 = r_2 = \dots = r_n$ , we have

$$\begin{aligned}s_1 &= r_1, s_2 = r_2 + \alpha_1\sigma\tau, s_3 = r_3 + \alpha_1\sigma\tau + \alpha_2\sigma\tau \\ &\dots\dots \\ s_n &= r_n + \sum_{i=1}^{n-1} \alpha_i\sigma\tau\end{aligned}$$

In the dynamic scheduling process, the  $n$  processor available times could be arbitrary. Thus, the analysis to get a closed-form solution for the completion time becomes very difficult. For this reason, in [6] we cast a homogenous cluster with different processor available times to a heterogeneous cluster model and then applied a DLT heterogeneous model to guide the task partitioning and to

derive a task execution time function. Although the intensive simulation results show that our previous approach does improve system performance, it cannot utilize all of the IITs ( $\sum_{i=1}^n p_i$ ).

In the remainder of this section, we first prove that IITs cannot always be completely eliminated. Then we focus on identifying the scenarios where we can fully utilize IITs.

**Theorem 2.** *It is not always possible to eliminate all IITs.*

**Constraint 1**  $r_i - r_{i-1} \geq \sigma\tau$ ,  $i = 2, 3, \dots, n$ .

Constraint 1 requires that the difference between available times of two successive processors to be no less than the total task data transmission time. This is a strong constraint. Later, we will propose another weaker constraint.

**Theorem 3.** *If Constraint 1 holds, Equations (3) and (4) result in an optimal partitioning of the load with no IITs.*

As we have mentioned, Constraint 1 is very strong. It is a sufficient but not a necessary condition for applying the optimal partitioning scheme. In some cases, Constraint 1 does not hold but the optimal partitioning approach is still applicable. Next we propose a weaker constraint to replace Constraint 1 and thus overcome its pessimism.

Note that for  $i = 2, 3, \dots, n$ , the start time of the  $i^{th}$  node will not be delayed if the data transmission time for the  $(i - 1)^{th}$  node is equal to or smaller than the difference between the two nodes' available times. Based on this observation, Constraint 2 is derived and Corollary 1 follows immediately from Theorem 1, since  $\forall i : 1 \leq i \leq n, s_i = r_i$  in this case.

**Constraint 2**  $\alpha_{i-1}\sigma\tau \leq r_i - r_{i-1}$ ,  $i = 1, \dots, n$ .

**Corollary 1.** *If Constraint 2 holds, Equations (3) and (4) result in an optimal partitioning of the load with no IITs.*

#### 4.5 Constraint-Based Algorithms

We first design algorithms based on Constraint 1. Figure 2 shows the general process of the schedulability test. After we detect IITs (i.e.,  $\exists r_i, r_j : r_i \neq r_j$ ), we first test whether Constraint 1 holds. If so, we repartition the task using our optimal partitioning approach presented in Section 4.3. Otherwise, we use the homogeneous-to-heterogeneous remodeling approach of [6] to repartition the task.

Various scheduling order policies can be integrated into our framework. In [6], we consider two such policies. One is FIFO, which is a common practice adopted by cluster administrators. The other is EDF, the Earliest-Deadline-First algorithm, which is a common real-time scheduling algorithm. The two algorithms we proposed in [6] are FIFO-DLT and EDF-DLT. They adopt different scheduling policies but apply the same partitioning and node assignment strategies. In

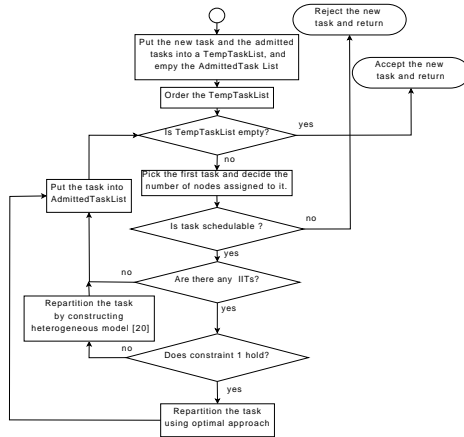


Fig. 2: Schedulability Test

this paper, by integrating our optimal partitioning approach with the two algorithms, we obtain two new algorithms: FIDO-I (Fifo Divisible load with Optimal partitioning) and EDDO-I (EDf Divisible load with Optimal partitioning).

The other two corresponding algorithms based on Constraint 2 are FIDO-II and EDDO-II. Unlike Constraint 1, Constraint 2 can only be verified after the task partition is determined because the value of  $\alpha_i$  cannot be known until we have the partitioning scheme. Thus, we first assume that Constraint 2 will be satisfied and use the optimal approach to partition the task. We check afterwards whether or not Constraint 2 really holds. If so, we are assured that the task partition is optimal. Otherwise, we fall back to applying the non-optimal partitioning strategy defined in [6].

## 5 Performance Evaluation

In this section, we first evaluate the proposed two sets of real-time scheduling algorithms: Set 1 = {FIDO-I, EDDO-I} and Set 2 = {FIDO-II and EDDO-II}. We compare the two algorithms in each set with the corresponding algorithms, FIFO-DLT and EDF-DLT, that we proposed in [6], which do not apply the optimal partitioning approach presented here. Second, we evaluate how communication cost will affect the performance of our approach.

### 5.1 Simulation Configurations

A discrete simulator is used to simulate a range of clusters that are compliant with the system model presented in Section 3. Three parameters,  $N$ ,  $\tau$  and  $\chi$  are specified for every cluster.

For a set of tasks  $T_i = (A_i, \sigma_i, D_i)$ ,  $A_i$ , the task arrival time, is specified by assuming that the interarrival times follow an exponential distribution with a

mean of  $1/\lambda$ . On the other hand, task data sizes  $\sigma_i$  are assumed to be normally distributed with the mean and the standard deviation equal to  $Avg\sigma$ . Task relative deadlines ( $D_i$ ) 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* [5]. 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  $DCRatio \times \mathcal{E}(Avg\sigma, N)$ . Thus, task relative deadlines are related 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:  $(N, \tau, \chi, 1/\lambda, Avg\sigma, DCRatio)$ .

To analyze the cluster load for a simulation, we use the metric *SystemLoad* [5]. It is defined as,  $SystemLoad = \frac{\mathcal{E}(Avg\sigma, N)}{\chi}$ , which is the 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  $(N, \tau, \chi, SystemLoad, Avg\sigma, DCRatio)$  is equivalent to specifying  $(N, \tau, \chi, 1/\lambda, Avg\sigma, DCRatio)$ , 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, a point on a curve corresponds to the average performance of ten simulations. For all ten runs, the same parameters  $(N, \tau, \chi, SystemLoad, Avg\sigma, DCRatio)$  are specified but different random numbers are generated for task arrival times  $A_i$ , data sizes  $\sigma_i$ , and deadlines  $D_i$ . For each simulation, the *TotalSimulationTime* is 10,000,000 time units, which is sufficiently long.

## 5.2 Advantages of Optimal Partitioning Strategy

As discussed in Section 4.4, the optimal partitioning strategy is applicable whenever Constraint 1 or Constraint 2 is satisfied. Therefore, we first analyze the frequency of cases in which Constraint 1 or Constraint 2 holds. If most of the time Constraint 1 or Constraint 2 does not hold, then the corresponding constraint-based algorithms are not going to be very useful. For convenience, we refer to the scenario when the scheduler detects IITs as S-IITs. In addition, if Constraint 1 holds, we refer to it as Opt-IITs, and if Constraint 2 holds, we refer to it as Opt-IITs-II.

To estimate the occurrence ratio of Opt-IITs and Opt-IITs-II to S-IITs, we conducted an experiment with the baseline system configuration  $N=16, \tau=1, \chi=100, Avg\sigma=200, DCRatio=2$  [6]. We executed the FIFO-DLT algorithm [6] and measured occurrences of S-IITs, Opt-IITs and Opt-IITs-II. The results are summarized in Table 1, where the last two columns present the occurrence ratio of Opt-IITs to S-IITs and the occurrence ratio of Opt-IITs-II to S-IITs. From the data we observe that in all cases 12%-16% of S-IITs are in fact Opt-IITs, and 14% -20% of S-IITs are Opt-IITs-II. This indicates there are a large number

of opportunities to apply our new approach to utilize IITs more efficiently and thus to improve system performance.

System Load	Number of S-IITs	Number of Opt-IITs	Number of Opt-IITs-II	Ratio of Opt-IITs to S-IITs	Ratio of Opt-IITs-II to S-IITs
0.1	131	21	22	16%	17%
0.2	262	31	36	12%	14%
0.3	665	105	113	16%	17%
0.4	1092	142	158	13%	14%
0.5	1927	229	289	12%	15%
0.6	2304	297	321	13%	14%
0.7	3560	493	601	14%	17%
0.8	4551	553	735	13%	16%
0.9	5661	757	980	14%	17%
1.0	6108	890	1203	15%	20%

Table 1: Occurrences of S-IITs and Opt-IITs. The S-IITs column indicates the number of times the scheduler detects IITs. The Opt-IITs column indicates the number of times Constraint 1 holds after the scheduler has detected IITs. The Opt-IITs-II column indicates the number of times Constraint 2 holds after the scheduler has detected IITs. The last two columns are the ratio of Opt-IITs to S-IITs and the ratio of Opt-IITs-II to S-IITs.

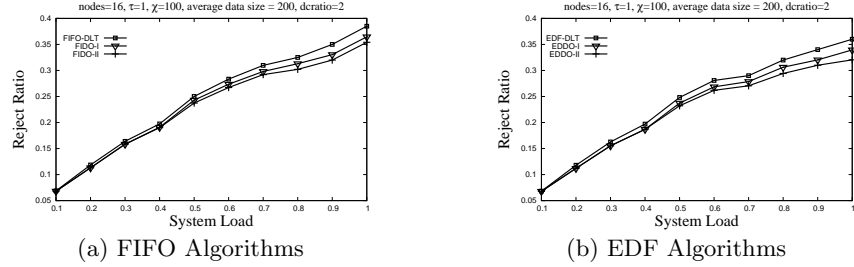


Fig. 3: Advantages of Optimal Partitioning

Figures 3a and 3b respectively show the comparison of algorithms FIDO-I and EDDO-I, FIDO-II and EDDO-II with their corresponding algorithms FIFO-DLT and EDF-DLT. From Figure 3a we observe that FIDO-I has a lower *Task Reject Ratio* than FIFO-DLT, which shows that applying the new partitioning approach leads to better performance. We also observe that FIDO-II has even better performance. These results confirm that because Constraint 2 is not as tight as Constraint 1, there are more chances to apply the optimal partitioning.

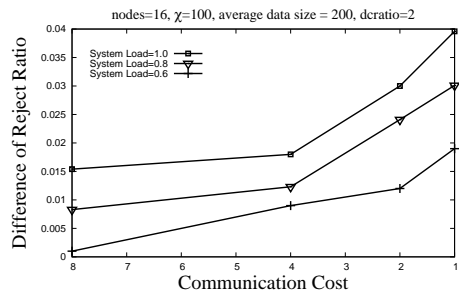


Fig. 4: Effects of Communication Cost

Figure 3b shows similar results. We conclude that it is beneficial to integrate our new partitioning approaches with real-time divisible load scheduling.

### 5.3 Effects of Communication Cost

In this section, we run simulations with decreasing values of  $\tau$  to study the effects of communication cost. With recent advancements in technology, networks are becoming faster and faster, and gigabits per second bandwidth has become commonplace. Moreover, the availability of thousands of wavelengths per fiber, and the development and deployment of all-optical switches and routers will result in further reductions in communication cost. For this reason, we investigate how decreases in communication costs will affect the performance of our approach.

For the simulation, we varied the values of  $\tau$  from 8 to 4, 2 and 1, while keeping the other parameters constant as the baseline configuration presented in Section 5.2. Each point in Figure 4 represents the difference in *Task Reject Ratios* for EDF-DLT and EDDO-II algorithms. The three curves represent the cases when *SystemLoad* is equal to 1.0, 0.8 and 0.6 respectively. We can see that, as the communication cost decreases, the difference in *Task Reject Ratios* becomes larger. This indicates that applying our new partitioning approach has more significant impact on system performance as the communication cost decreases. The reason is that  $\sigma\tau$  becomes smaller when communication cost decreases, and thus both Constraints 1 and 2 become less restrictive. Consequently, we have more opportunities to apply the optimal partitioning and better utilize IITs.

## 6 Conclusion

In this paper, we address the Inserted Idle Times (IITs) problem in the context of real-time divisible load scheduling [5]. Two contributions are made. First, we propose two constraints for the existence of the optimal partitioning that can fully utilize IITs. Second, we integrate this approach with our previous work [6] to develop new real-time scheduling algorithms that utilize IITs. Simulation results show that our approach makes use of IITs to a larger extent and significantly improves the system performance. Currently, we are working on expanding our approach to show that by adopting multi-round scheduling [13], we can further improve the IITs utilization and the system performance.

## References

1. 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/>)
2. ATLAS (AToroidal LHC Apparatus) Experiment, CERN (European Lab for Particle Physics): Atlas web page. (<http://atlas.ch/>)
3. Swanson, D.: Personal communication. Director, UNL Research Computing Facility (RCF) and UNL CMS Tier-2 Site (2005)
4. Robertazzi, T.G.: Ten reasons to use divisible load theory. *Computer* **36**(5) (2003) 63–68
5. Lin, X., Lu, Y., Deogun, J., Goddard, S.: Real-time divisible load scheduling for cluster computing. In: 13th IEEE Real-Time and Embedded Technology and Application Symposium, Bellevue, WA (2007) 303–314
6. Lin, X., Lu, Y., Deogun, J., Goddard, S.: Real-time divisible load scheduling with different processor available times. Technical Report TR-UNL-CSE-2007-0013, University of Nebraska-Lincoln (2007)
7. Lifka, D.A.: The anl/ibm sp scheduling system. In: IPPS '95: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, London, UK, Springer-Verlag (1995) 295–303
8. Anderson, J.H., Srinivasan, A.: Pfair scheduling: beyond periodic task systems. In: RTCSA '00: Proceedings of the Seventh International Conference on Real-Time Systems and Applications (RTCSA'00), Cheju Island, South Korea (2000) 297–306
9. Lee, W.Y., Hong, S.J., Kim, J.: On-line scheduling of scalable real-time tasks on multiprocessor systems. *Journal of Parallel and Distributed Computing* **63**(12) (2003) 1315–1324
10. Yeo, C.S., Buyya, R.: A taxonomy of market-based resource management systems for utility-driven cluster computing. *Softw. Pract. Exper.* **36**(13) (2006) 1381–1419
11. Amir, Y., Awerbuch, B., Barak, A., Borgstrom, R., Keren, A.: An opportunity cost approach for job assignment in a scalable computing cluster. *IEEE Trans. on Parallel and Distributed Systems* **11**(7) (2000) 760–768
12. Barsanti, L., Sodan, A.C.: 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 (2006) 115–140
13. Bharadwaj, V., Robertazzi, T.G., Ghose, D.: *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press, Los Alamitos, CA (1996)
14. Veeravalli, B., Ghose, D., Robertazzi, T.G.: Divisible load theory: A new paradigm for load scheduling in distributed systems. *Cluster Computing* **6**(1) (2003) 7 – 17
15. Dertouzos, M.L., Mok, A.K.: Multiprocessor online scheduling of hard-real-time tasks. *IEEE Trans. Softw. Eng.* **15**(12) (1989) 1497–1506
16. Manimaran, G., Murthy, C.S.R.: An efficient dynamic scheduling algorithm for multiprocessor real-time systems. *IEEE Trans. on Parallel and Distributed Systems* **9**(3) (1998) 312–319
17. Ramamritham, K., Stankovic, J.A., Shiah, P.f.: Efficient scheduling algorithms for real-time multiprocessor systems. *IEEE Trans. on Parallel and Distributed Systems* **1**(2) (1990) 184–194