Real-Time Scheduling of Divisible Loads in Clusters to Handle Estimated Execution Time Inaccuracies

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

Abstract
Quality of Service (QoS) provisioning for divisible loads in clusters, enabled using real-time scheduling theory, is based on an important assumption: the scheduler knows the execution time of every task in the workload. Information from production clusters, however, shows that user estimated task execution times are often inaccurate. In this paper, we present an approach to address this problem, where a feedback mechanism is developed to overcome the inaccuracies in user estimation of task execution time. Besides QoS provisioning of divisible loads in clusters, the proposed algorithm achieves low task reject ratio and high system utilization. Simulation evaluation shows that the new algorithm outperforms the previous approach.

1 Introduction
Arbitrarily divisible or embarrassingly parallel workloads can be partitioned into an arbitrarily large number of load fractions, and are quite common in bioinformatics as well as high energy and particle physics. For example, the CMS (Compact Muon Solenoid) [10] and ATLAS (AToroidal LHC Apparatus) [5] projects, associated with LHC (Large Hadron Collider) at CERN (European Laboratory for Particle Physics), execute cluster-based applications with arbitrarily divisible loads. As such applications become a major type of cluster workloads [23], providing QoS to arbitrarily divisible loads becomes a significant problem for cluster-based research computing facilities like the U.S. CMS Tier-2 sites [25].

Real-time divisible load scheduling is a well researched area [15, 20, 18, 21, 8, 9]. Previous work is, however, based on an important assumption: the scheduler knows the execution time of every task in the workload. If this information is inaccurate, scheduling decisions could be highly inefficient. Estimation of task execution time is a hard problem [28]. Although much work has been done to improve the estimation, there are always uncertainties in task execution times. In distributed systems, this problem becomes even harder because a task might be executed on multiple processors and time for data communication also has to be considered [11]. Usually, the estimated task execution time is provided to the scheduler along with other task parameters. In most cases, this estimation is the worst-case execution time, usually based on the user knowledge of the task. A user who works with clusters tends to overestimate this value to avoid his/her job being killed “just in case” the job runs longer.

We studied a year’s worth of logs for production jobs submitted to the Red and PrairieFire clusters 1 at the University of Nebraska-Lincoln (UNL). From this study, we found that among the submitted jobs, 95% to 97% execution times are overestimated. The actual average execution times are just 9% and 18% of the estimated execution times, respectively for Red and PrairieFire. This shows gross overestimation of job execution times by the user. In Table 1, we show the job statistics of Red and PrairieFire clusters. Statistics show that only 3% to 5% of total number of execution times are underestimated and these jobs exceed execution beyond their estimated times. According to current practice, most of the jobs exceeding their estimated execution times are killed. The system logs show that about 91% of such jobs on PrairieFire and 98% on Red, are killed.

QoS provisioning for divisible loads involves three components: an admission controller that decides to accept or reject an incoming task, a scheduler that schedules and partitions admitted tasks into subtasks, a dispatcher that sends the subtasks to the processors at their scheduled times. The scheduler makes decisions based on task parameters, such as execution time and deadline. If a task is admitted, its subtasks are placed into a queue and later dispatched by the dispatcher. One problem with this model is that once the schedule for a task (and its subtasks) is set, it is not changed. Even if a sufficient number of nodes become available before the scheduled task start time, they remain idle. The processing capability of the cluster is, therefore, wasted. Another problem is that the scheduler does not know how long a task will run after it runs beyond its allocated time. So, such tasks are generally killed to enforce the schedule. Task killing is, however, undesirable because the time the cluster has already spent on killed tasks is completely wasted.

The main contribution of this paper is to enhance the real-time scheduling of divisible tasks by better handling inaccuracies in estimation of the task execution times. A new algorithm called FDLS (Feedback Divisible Load Scheduling) is developed. In this algorithm, execution time inaccuracies are handled by dynamically updating the schedule of tasks waiting for execution. This is accomplished by a feedback module, which is integrated with EDF-DLT algorithm to develop FDLS [18].

1Red and PrairieFire are respectively 215-node and 138-node production-mode LINUX clusters.
Our feedback module can better utilize the idle time created by early completion of underestimated jobs. In addition, FDLS judiciously reduces task killing. Task real-time constraints are guaranteed as long as their execution times are not underestimated. Some tasks are allowed to exceed their allocated execution times, but it is guaranteed that other tasks will not miss their deadlines or get rejected as a result.

Simulations show that with execution time uncertainties, the new algorithm outperforms EDF-DLT, one of the best performing algorithms for real-time divisible load scheduling in clusters [18, 19]. Moreover, the new algorithm provides better Task Reject Ratio and considerably improves System Utilization.

The rest of the paper is organized as follows. In Section 2, we discuss related work. In Section 3 we present the task and system models. Description of the feedback scheduling algorithm is presented in Section 4. We evaluate the algorithm performance in Section 5 and conclude the paper in Section 6.

2 Related Work

Many researchers have investigated the general problem of scheduling workloads for clusters of computers [29, 4, 3, 22]. The most closely related work to ours is the scheduling of "scalable tasks" [15] or "moldable jobs" [6]. To the best of our knowledge, however, only [15] and [14] have considered QoS support for such loads.

In [20] we investigated real-time cluster-based divisible load scheduling and proposed several algorithms. When scheduling a parallel job, however, if a sufficient number of processors are not immediately available, the job waits for additional processors. This essentially leads to a waste of processing power as some processors sit idle waiting for a job to start execution. This is a system inefficiency that we refer to as the Inserted Idle Times (IITs) problem.

Historically, backfilling algorithms [16] have been proposed to address idle times in a schedule, where small jobs could be moved ahead and run on processors that would otherwise remain idle. A mechanism to utilize processor idle-times, also called fragments, was investigated in [15], wherein a task is assigned a larger number of nodes to utilize more processing power. Complementary to that approach, we proposed a real-time divisible load scheduling approach called EDF-DLT that utilizes IITs [18]. EDF-DLT enables a task to utilize a processor as soon as it becomes available. While results in [15] showed that the performance improvement of their approach was negligible, EDF-DLT was shown to lead to a much better performance — outperforming both backfilling algorithms and the approach developed in [15].

We proved in [19], however, that it is not always possible to eliminate all IITs, except when certain constraints are met. An enhancement to EDF-DLT algorithm was then presented that is able to improve performance under these circumstances. These constraints, however, are entirely dependent on knowing the precise workload demand of a task before it executes. As stated previously, this rarely, if ever, occurs in practice. Thus, while the approach proposed in [19] is important and of theoretical significance, its full benefit will not be realized until better execution time estimates are available. In this paper, we take a different approach to enhancing EDF-DLT algorithm that we believe has tremendous potential for improving cluster scheduling. We evaluate the performance of our approach via simulations.

3 Task and System Models

Task Model. A divisible task $T_i$ is denoted by the tuple $T_i = (A_i, \sigma_i, D_i)$ where $A_i$ is arrival time, $\sigma_i$ is data size and $D_i$ is relative deadline of the task. A workload consists of a set of independent tasks. A task is arbitrarily divisible if it can be partitioned into a set of arbitrarily small size subtasks. We use the vector $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ to denote the data partition of a task when $n$ processing nodes are assigned to such a task. $\alpha_i$ is the data fraction allocated to the $i^{th}$ subtask, i.e., $\alpha_i \sigma$ units of data is assigned to subtask $i$. We have $0 < \alpha_i \leq 1$ and $\sum_{i=1}^{n} \alpha_i = 1$.

System Model. The system consists of a cluster with a head node, denoted $P_0$, connected to $N$ processing nodes, denoted $P_1, P_2, \ldots, P_N$, via a switch. Every processing node in the cluster has the same computational capability and the same bandwidth on its link to the head node. The head node does not participate in the computation but takes the role of the admission controller, the scheduler and the dispatcher. By assumption, data transmission from the head node cannot be done in parallel and only one processing node can receive data from the head node at a time.

Applying divisible load theory, transmission and computation time of a task is represented by a linear model. The transmission and computation time of $\sigma$ data units is given by $\sigma C_{ms}$ and $\sigma C_{ps}$. $C_{ms}$ represents the time to transmit a unit of workload from the head node to a processing node. $C_{ps}$ represents the time to compute a unit of workload on a processing node.

4 Algorithms

This section presents the Feedback Divisible Load Scheduling (FDLS) algorithm for scheduling real-time divisible loads with inaccurate execution time estimates. To develop our feedback algorithm, we adapt EDF-DLT algorithm in [18]. The primary

<table>
<thead>
<tr>
<th>Type of jobs</th>
<th>Red</th>
<th>PrairieFire</th>
<th>Red</th>
<th>PrairieFire</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jobs that finish on time</td>
<td>0</td>
<td>0</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Jobs with overestimated execution time</td>
<td>188545</td>
<td>26193</td>
<td>96.87%</td>
<td>95.03%</td>
</tr>
<tr>
<td>Jobs with underestimated execution time</td>
<td>6103</td>
<td>1370</td>
<td>3.13%</td>
<td>4.97%</td>
</tr>
<tr>
<td>Underestimated jobs that are killed</td>
<td>5963</td>
<td>1240</td>
<td>3.06%</td>
<td>4.49%</td>
</tr>
<tr>
<td>Total</td>
<td>194648</td>
<td>27563</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 1. Job statistics from two real clusters
idea of EDF-DLT is to dispatch a subtask at the estimated available time of a processing node, so that the idle time in a cluster node can be better utilized. Recall that $P_1, P_2, \ldots, P_n$ denote the $n$ processing nodes. Assume that node $P_i$ starts processing task $T$ at time $r_i$, for $i = 1, 2, \ldots, n$. By $r_i$, we denote 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 later. The $n$ nodes are ordered by their available times: $P_1$ has the earliest time $r_1$ and $P_n$ has the latest at time $r_n$.

When a task $T$ arrives, the scheduler calculates the minimum number of nodes to be assigned to $T$ to guarantee its deadline [18]

$$\hat{n}^{\text{min}} = \left\lfloor \frac{\ln \gamma}{\ln \beta} \right\rfloor \tag{1}$$

where

$$\gamma = 1 - \frac{\sigma C_{ms}}{A + D - r_n} \tag{2}$$

and

$$\beta = \frac{C_{ps}}{C_{ms} + C_{ps}}. \tag{3}$$

According to divisible load theory (DLT), the optimal execution time is obtained when all nodes allocated to a task complete their computation at the same time [26]. In [18], we propose EDF-DLT algorithm that utilizes inserted idle times (IITs). Following the algorithm, we can schedule a real-time divisible load with different node available times and finish them at approximately the same time. As shown in [18], the resultant task execution time, denoted by $\hat{E}$, is

$$\hat{E}(\sigma, n) = \sigma C_{ms} + \frac{\prod_{j=2}^{n} X_j}{1 + \sum_{i=2}^{n} \prod_{j=2}^{i} X_j} \sigma C_{ps} \tag{4}$$

where

$$X_i = \frac{\frac{\sigma C_{ps}}{\varepsilon + r_n - r_{i-1}}}{C_{ms} + \frac{\sigma C_{ps}}{\varepsilon + r_n - r_i}}, \text{ for } i = 2, 3, \ldots, n \tag{5}$$

and

$$\varepsilon = \frac{1 - \beta}{1 - \beta^n} \sigma(C_{ms} + C_{ps}). \tag{6}$$

The desired data distribution vector is given as

$$\alpha_1 = \frac{1}{1 + \sum_{i=2}^{n} \prod_{j=2}^{i} X_j} \tag{7}$$

and

$$\alpha_i = \frac{\prod_{j=2}^{i} X_j}{1 + \sum_{i=2}^{n} \prod_{j=2}^{i} X_j}, \text{ for } i = 2, 3, \ldots, n. \tag{8}$$

The results from [18] show that EDF-DLT is one of the best known scheduling algorithms for real-time divisible loads in clusters. However, this algorithm assumes that the task execution time estimate is accurate. If the actual computation time does not match the user estimate, a task either finishes earlier or runs past the estimated execution time. Since there is no feedback mechanism incorporated in the aforementioned algorithm, the scheduler has no means of knowing about these situations. This leads to idle time that is not utilized or to a task being killed because its execution exceeds allocated time.

We propose FDLS, a new DLT-based scheduling algorithm with a feedback mechanism, to address these deficiencies. Its goal is to better utilize the processing nodes and minimize the number of tasks that are killed. We use the following definitions:

- A task is said to “underrun” if its actual execution time is smaller than the estimated time. Most of the tasks on real-life clusters fall into this category.

- A task is said to “overrun” if its execution time is larger than the estimated time. While less common, overruns are very expensive in terms of wasted resources if tasks are killed.

In FDLS, we record and use the historical information of the real execution times of tasks to derive the estimated execution time in the admission control of the algorithm. Estimated execution time is computed from the estimated computation cost, and the known communication cost, as in Equation (4). When a task finishes, we compute the estimated computation cost ($C_{ps}[*]$) as in Equation (9). In this estimation, we do not consider the overrun task execution times because we have the option to kill them when resources are needed by other tasks.

$$C_{ps}[i] = P_U[i] * C_{ps} * K[i] + (1 - P_U[i]) * C_{ps} \tag{9}$$

where $C_{ps}$ is the initial estimated computation cost of the workload. $P_U[i]$ is the percentage of underrun tasks encountered so far and $K[i]$ is the average underrun amount of these tasks. They are computed as below:

$$P_U[i] = \frac{\Delta[i] + 1}{A[i]} \tag{10}$$

$$K[i] = \frac{\Delta[i] * K[i] + C_{ps}[i]}{C_{ps}} \tag{11}$$

$$\Delta[i] = \Delta[i] + 1 \tag{12}$$

where $\Delta[i]$ is the number of underrun tasks so far, $A$ is the number of arrived tasks so far, $C_{ps}[i]$ is the computation cost of the task that has just finished.
The general process flow of FDLS algorithm is shown in Pseudocode 1. The NewTaskEvent is triggered when a task arrives. We use the function AC-GS (Admission Control and Generate Schedule) to check if we can accept the task or not. If it is accepted, this function partitions the data and generates the schedule for the task. The AC-GS function represents an improvement of the SchedulabilityTest function of EDF-DLT algorithm in [18].

Due to the feedback mechanism, the system is able to detect and handle the two events: OverrunTimerEvent and TerminationEvent. The first event is triggered at the expected completion time of a running subtask. The second event is triggered when a subtask finishes its execution. The solutions to handle these two events are described below.

When a sub task needs to be dispatched, the system invokes the DispatchTask function. This function sends a subtask in the queue to a processing node in the cluster. After dispatching a subtask, the system will reset the DispatchTimer to the time when the next subtask in the queue should be dispatched.

**Handling Overrun and Underrun Tasks**

Since the scheduler is not clairvoyant, it does not know if a task underruns or overruns until its subtasks finish. Therefore, if a task overruns, it is not possible for the scheduler to estimate the task completion time. The nodes occupied by overrun tasks are considered to be blocked. An overrun task can therefore severely affect the scheduling of other tasks and results in deadline misses and task rejections.

A common practice in real-life clusters is to kill overrun tasks; EDF-DLT algorithm also uses such an approach to prevent overrun tasks from causing real-time performance problems. However, killing an overrun task is costly because the time spent on that task is wasted and the task must be re-submitted and re-executed later. Thus, in our algorithm, we aim not to kill overrun tasks as long as they have no negative effect on the system performance and schedulability of other tasks.

The HandleOverrun function is presented in Pseudocode 2. Assume that an overrun task occurs at time $t$, we need to gather the following information in order to handle the situation

$N_{\text{block}}$: Number of nodes that are blocked by overrun tasks.

$N_T$: Number of subtasks waiting to be dispatched.

$N_{\text{idle}}$: Number of idle nodes.

It is noted that since at time $t$ there is at least one overrun task, $N_{\text{block}} > 0$, $N_T > 0$ and $N_{\text{idle}} > 0$. Based on $N_T$ and $N_{\text{idle}}$, we evaluate the latest time $t'$ that blocked nodes have to be dispatched to enforce the current schedule. There are two cases

- Case 1: $0 \leq N_T \leq N_{\text{idle}}$: In this case, there are subtasks that need to be dispatched at time $t$, and sufficient nodes are available. Therefore overrun tasks can continue their computation. However, since the overrun nodes may be needed by incoming new tasks, these nodes are marked as AvailableButOccupied.

Pseudocode 2 HandleOverrun()

```
1: newEndTime ← 0
2: nOverrunNodes ← number of nodes that have overrun jobs
   //this is $N_{\text{block}}$
3: nAvailableNodes ← number of available nodes //this is $N_{\text{idle}}$
4: nWaitingSubtask ← number of subtasks waiting in the queue
   //this is $N_T$
5: if nWaitingSubtask ≤ nAvailableNodes then
6:   newEndTime ← ∞
7: else
8:   subtaskQueue ← subtask waiting queue
9:   sort subtaskQueue by subtask’s start time
10:  k ← nWaitingSubtask – nAvailableNodes
11:  newEndTime ← start time of the $k^{th}$ subtask in queue
12:  end if
13: if newEndTime == currentTime then
14:   kill overrun jobs
15:  set status of overrun nodes to available
16: else if newEndTime == ∞ then
17:  set status of overrun nodes to AvailableButOccupied
18: else
19:  set new available time of overrun nodes to newEndTime
20: end if
```

- Case 2: $N_T > N_{\text{idle}}$: In this case, a sufficient number of nodes are not available. However, not all subtasks start occupying the processing nodes at the same time (due to the data transmission). Therefore, if we order the subtasks in increasing order of their start time, we can let the overrun tasks continue their execution until the $k^{th}$ subtask starts, with $k = N_T – N_{\text{idle}}$.

As opposed to the overrun case, the solution for underrun tasks is relatively straightforward. The system knows immediately when a task underruns because of the feedback mechanism, i.e., TerminationEvent is triggered before the expected completion time of the task. Therefore, nodes finishing time could be updated, and if there is a pending task in the dispatching queue, this task is dispatched immediately.

**5 Performance Evaluation**

In this section, FDLS algorithm is evaluated via simulations, where we compare FDLS algorithm with one of the previously best known approaches for divisible load scheduling: EDF-DLT [18].

Our main evaluation metric is Task Reject Ratio, which is defined as the number of rejected tasks and killed tasks over the total number of arrived tasks. Besides, we also use System Utilization to evaluate the algorithms. Our simulations reflect the task and system models presented in Section 3. A cluster is configured by three parameters: $N$, $C_{\text{ms}}$ and $C_p$, where $N$ is the number of nodes of the cluster.

**Simulation workload generation**: The workload is generated following the same approach as described in [20, 18] and due to the space limitation, we choose not to repeat the details here. A metric $\text{SystemLoad} = \varepsilon(Avg\sigma, N)\lambda$ is defined to analyze how loaded a cluster is for a simulation, where $\lambda$ is the average task arrival rate of the cluster, $Avg\sigma$ is the average task data size,
and \( \varepsilon(\text{Avg} \sigma, N) \) is the execution time of running a task of size \( \text{Avg} \sigma \) on \( N \) nodes (see Equation (6) for \( \varepsilon \)'s calculation).

We introduce four parameters \( P_O, P_U, k_O, k_U \) to model the overrun and underrun tasks in the workload. The first two parameters, \( P_O, P_U \), are the percent of overrun and underrun tasks in the workload. For example, if \( P_O = 20\% \), 20 out of 100 tasks have \( Cps \) larger than estimated. The other parameters, \( k_O \) and \( k_U \), are overrun and underrun scale factors applied to \( Cps \). For example, if \( k_O = 120\% \) the actual \( Cps \) are 120\% of the estimate. Different combinations of \( P_O(P_U) \) and \( k_O(k_U) \) can be used to study a broad range of workloads.

**Simulation design:** In our simulations, we use the following parameters: \( N = 16, Cms = 1, Cps = 100, \text{Avg} \sigma = 200, DCRatio=2 \quad (\text{Figure 2, 4}) \). \( SystemLoad \) changes from 0.1 to 1.0. The \( SystemLoad \) is computed from the user estimated execution time of tasks in the workload.

\[
\begin{array}{c|c|c}
\text{Prairie Fire} & \text{Red} \\
\hline
P_U & 95\% & 97\% \\
\hline
k_U & 20\% & 70\% \\
\hline
P_O & 5\% & 3\% \\
\hline
k_O & 150\% & 120\%
\end{array}
\]

**Table 2. Simulation parameters for the workloads**

We construct experiments using the workload characteristics of Red and PrairieFire, the two production clusters at the University of Nebraska-Lincoln. Simulation parameters are shown in Table 2. We compare three versions of algorithms: EDF-DLT, FDLS and EDF-DLT best case, in which task execution times are accurately known in advance.

Results of two workloads are shown in Figure 1, 3, 2, 4. The analysis is conducted in terms of **Task Reject Ratio** (Figure 1, 3) and **System Utilization** (Figure 2, 4). We see that EDF-DLT best case outperforms EDF-DLT and FDLS algorithms in all metrics. However, FDLS always performs better than EDF-DLT in terms of both **Task Reject Ratio** and **System Utilization**.

**6 Conclusion**

Inaccuracies in task execution times lead to highly inefficient schedule and severely affect the system performance. It is a particularly difficult problem for real-time scheduling of divisible loads in clusters. From the logs of real-life production clusters, we find that execution times of a large percentage of tasks are overestimated and a small percentage are underestimated. Overestimation of execution times leads to processor idle time and underestimation forces tasks to be killed or miss their deadlines. In this paper, we address the problem of inaccuracies in the user estimation of task execution times in the context of real-time divisible load scheduling. We present an algorithm to minimize the effect of estimated execution time inaccuracies, where a strategy is developed to identify and handle overrun and underrun divisible tasks. We enforce the QoS and real-time constraints of the system by integrating the feedback mechanism into the scheduling algorithm. Simulations show that our algorithm significantly improves system performance with different levels of uncertainties in task execution times.
References


