Resource-Aware Distributed Scheduling Strategies for Large-Scale Computational Cluster/Grid Systems

Sivakumar Viswanathan, *Student Member*, *IEEE*,
Bharadwaj Veeravalli, *Senior Member*, *IEEE*, and Thomas G. Robertazzi, *Fellow*, *IEEE*

Abstract—In this paper, we propose distributed algorithms referred to as *Resource-Aware Dynamic Incremental Scheduling* (RADIS) strategies. Our strategies are specifically designed to handle large volumes of computationally intensive arbitrarily divisible loads submitted for processing at cluster/grid systems involving multiple sources and sinks (processing nodes). We consider a real-life scenario, wherein the buffer space (memory) available at the sinks (required for holding and processing the loads) varies over time, and the loads have deadlines and propose efficient "pull-based" scheduling strategies with an admission control policy that ensures that the admitted loads are processed, satisfying their deadline requirements. The design of our proposed strategies adopts the divisible load paradigm, referred to as the divisible load theory (DLT), which is shown to be efficient in handling large volume loads. We demonstrate detailed workings of the proposed algorithms via a simulation study by using real-life parameters obtained from a major physics experiment.

Index Terms—Divisible loads, grid computing, cluster computing, buffer constraints, processing time, deadlines.

1 Introduction

Scheduling in grid systems is a challenging task, as it involves coordinating multiple computational sites for resource sharing and scheduling in an efficient manner. These systems are exclusively meant for handling large volumes of computational loads such as data generated in the high-energy nuclear physics experiments [10], bioinformatics [11], astronomical computations, etc. These applications demand new strategies for collecting, sharing, transferring, and analyzing the data. This relatively new era of computing, referred to as grid computing [19], expands collaborations and intense data analysis, coupled with increasing computational and networking capabilities.

In general, a grid computing environment is comprised of large groups of diverse geographically distributed resources (clusters) that are collected into a virtual computer for high-performance computation. Grid computing creates middleware and standards to function between these computers and networks. It allows full resource sharing among individuals, research institutes, and corporate organizations. It dynamically allocates the idle computing capability to the needed users at remote sites. The large number and diverse nature of these computing resources and their users pose a significant challenge to efficiently schedule the loads and utilize the resources. The motivation for our work stems from the challenges in managing and utilizing computing resources in grids as efficiently as possible. To date, there has been little or no work on resource-aware distributed dynamic scheduling of large-volume divisible loads with deadline requirements on a cluster node in a grid environment.

In this work, we propose Resource-Aware Dynamic Incremental Scheduling (RADIS) strategies for situations wherein processing for several divisible (partitionable) loads need to be completed within their respective deadline requirements, while the processing nodes have finite capacity constraints. Divisible loads are a class of loads that require homogeneous processing and can be partitioned into arbitrary smaller fractions [12]. These load portions, which bear no dependence relationships among themselves, can then be assigned to individual nodes for processing. We provide a detailed analysis of our algorithms and demonstrate their performance by using a simulation study, with real-life parameters derived from high-energy nuclear physics experiments discussed in [10]. The analytical flexibility offered by the divisible load theory (DLT) is thoroughly exploited to design resource-conscious algorithms that make the best use of the available resources in a cluster. We employ both interleaving and noninterleaving techniques to process tasks (jobs) that are admitted into the system and discuss their usefulness. Our systematic design clearly elicits the advantages offered by our strategies. The paper is organized as follows: In Section 2, we provide the research background and related work for grid and cluster scheduling and DLT. In Section 3, we formalize the multisource and multisink problem at a cluster node in a grid system. In Section 4, we discuss our

- *S. Viswanathan and B. Veeravalli are with the Computer Networks and Distributed Systems (CNDS) Laboratory, Department of Electrical and Computer Engineering, The National University of Singapore, Singapore. E-mail: {g0306272, elebv}@nus.edu.sg.*
- *T.G. Robertazzi is with the Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY 11794. E-mail: tom@ece.sunysb.edu.*

Manuscript received 6 Sept. 2006; accepted 12 Dec. 2006; published online 1 Feb. 2007.
Recommended for acceptance by R. Thakur.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0277-0906.
Digital Object Identifier no. 10.1109/TPDS.2007.1073.

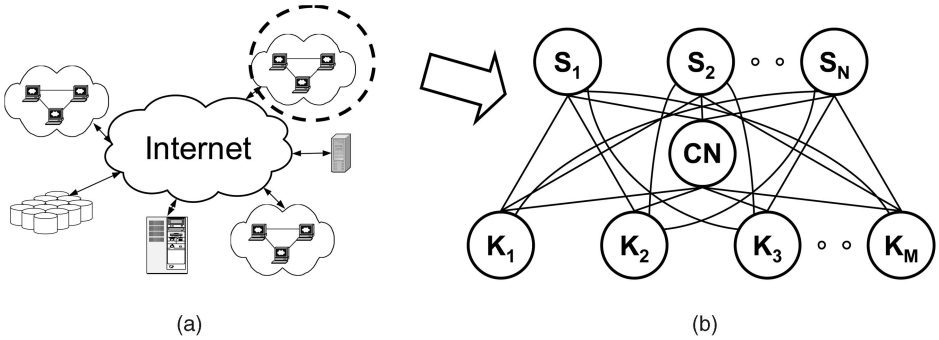

Fig. 1. Grid computing system. (a) Grid infrastructure. (b) Abstract overview of a cluster comprising sources and sinks with a coordinator node (*CN*).

scheduling strategies. In Section 5, we discuss the performance of our algorithms and their workings and highlight their advantages with a simulation study. In Section 6, we provide the conclusions.

2 Related Work

In this section, we present some of the works that are relevant to the problem addressed in this paper. For divisible loads, research since 1988 has established that optimal allocation/scheduling of a divisible load to processors and links can be solved through the use of a very tractable linear model formulation, referred to as DLT. The contributions in [23], [24] looked at the problem of seeking optimal solutions for scheduling "large-grained" computations on loosely coupled processor systems. The study by Agrawal and Jagadish [23] focused on single-level tree architecture, whereas the one by Cheng and Robertazzi [24] considered daisy-chained systems. DLT is rich in such features as easy computation, a schematic language, equivalent network element modeling, results for infinite-sized networks, and numerous applications. This linear theory formulation opens up attractive modeling possibilities for systems incorporating communication and computation issues, as in parallel, distributed, and grid computing. Here, the optimality, involving solution time and speedup, is defined in the context of a specific scheduling policy and interconnection topology. The linear model formulation usually produces optimal solutions through linear equation solutions. In simpler models, recursive algebra also produces optimal solutions. The model can take into account heterogeneous processor and link speeds, as well as relative computation and communication intensity.

DLT can model a wide variety of approaches with respect to load distribution (sequential or concurrent), communications (store and forward and virtual cut-through switching), and hardware availability (the presence or absence of front-end processors). Front-end processors allow a processor to both communicate and compute simultaneously by assuming communication duties. In addition to the monograph [22], a survey on the results until 2003 has been published in [12]. DLT has been proven to be remarkably flexible. The DLT model allows analytical tractability to derive a rich set of results regarding several important properties of the proposed strategies and to analyze their performance. DLT also offers an exciting opportunity to optimally schedule multiple divisible loads in grid computing. The usefulness of DLT in terms of its applicability is demonstrated in several real-life applications, for example, parallel video encoding [3], image processing [14], large matrix computations [15], and database applications [17], [18].

Moges et al. [7] studied divisible load scheduling strategy, with communication delays and two source nodes. Kim [13] has proposed a mathematical model in which simultaneous communication to several nodes is carried out. This model suits a cluster node in our grid system infrastructure. A very directly relevant material to the problem addressed in our paper is in [10], where Wong et al. use an incremental recursive strategy (modified incremental balancing strategy (IBS) algorithm) for offline scheduling of multiple loads in a grid environment. Recently, memory-constrained problem formulations for grid systems are also considered. Wu and Sun [8] studied memory-conscious task scheduling for grid systems. Kim and Weissman [9] presented a genetic algorithm approach for decomposable data processing on large-scale data grids. Marchal et al. [4] considered scheduling divisible loads for generic large-scale platforms. Another study that may be useful in the cluster systems context is of Ghose et al. [2], wherein time varying speeds of links and processors in the network are considered in the modeling to evolve an adaptive load distribution strategy. Beaumont et al. [5] discussed some open-ended problems and issues pertaining to divisible load scheduling. We refer astute readers to the papers mentioned above for an up-to-date look at the literature related to the problem addressed in this paper.

3 Problem Formulation

A generic grid computing system infrastructure considered here comprises a network of supercomputers and/or a cluster of computers connected by local area networks, as shown in Fig. 1a, having different computational and communication capabilities. We consider the problem of scheduling large-volume loads (divisible loads) within a cluster system, which is part of a grid infrastructure. We envisage this cluster system as a *cluster node* comprising a set of computing nodes. Communication is assumed to be predominant between such cluster nodes and is assumed to be negligible within a cluster node. The underlying computing system within a cluster can be modeled as a fully connected bipartite graph comprising *sources*, which have

computationally intensive loads to be processed (very many operations are performed on them) and computing elements, called *sinks*, for processing loads (data), as shown in Fig. 1b. This represents the fact that each source can schedule its load on all the sinks.

In real-life situations, one of the practical constraints is satisfying the deadline requirements of the loads (arriving in real time from multiple source nodes) to be processed while taking into account the availability of the buffer (memory) resources at the sink nodes, since the memory available at the processing nodes to store the received load and process them is limited. We consider these combined influences in our proposed algorithm. We employ a "pull-based" approach in the design of our scheduling strategy, wherein the sinks schedule the competing sources, depending on the availability of the resources for processing.

Now, we shall formally define the problem that we address. We consider a cluster node in a grid system, comprising N source nodes denoted as $S_1, S_2, \ldots, S_N$ and M sink nodes denoted as $K_1, K_2, \ldots, K_M$. Each source S_i has a load L_i to be processed. In our model, a master node is assumed to coordinate the activities within a cluster. The master node estimates the load distribution and does admission control for the sources. We refer to this master node simply as a *coordinator node (CN)*, and without loss of generality, we assume that any node within a cluster can be elected as the CN based on leader election algorithms [20].

As shown in Fig. 1b, there are direct links (which may be virtual) from all source and sink nodes to *CN*. In this paper, we adopt a simultaneous load distribution model, as proposed in [21], in which all sources (sinks) can send (receive) load fractions to all the sinks (from all the sources) simultaneously. Also, following Kim's model [13], we assume that the communication time delay is negligibly smaller than the computation time, owing to high-speed links within a cluster node, so that no sink starves for load and that all sinks could start computing as they receive the loads from the sources.

The objective of this study is to schedule and process the loads among M sink nodes, rendering finite buffer capacities such that their *processing time*, defined as the time instant when all the M sinks have completed processing the loads is a minimum. As with real-life situations, we consider the availability of buffer space as a time-varying quantity in our formulation. Also, our objective is to minimize the scheduling-related communication overheads in the system. The CN obtains the information about the available memory capacities and computing speeds from the sinks, as well as the size and deadline requirements of the loads from the sources. The CN then computes the parameters required by the sinks for scheduling and broadcasts them to all of the sinks. The sink nodes determine the amount of load fractions to be received from the source nodes based on the scheduling parameters received from the CN. Thus, in this study, all the proposed schemes are distributed scheduling strategies. The sources, upon receiving the requests from the sinks, shall send their load to all sinks concurrently.

Our RADIS strategies proposed in this paper are a generalization of the modified IBS algorithm [10], tuned to consider dynamic arrival of loads with deadline constraints. We also propose admissibility criteria to handle such dynamic loads. We now present the list of notations, definitions, and terminology that will be used throughout the paper:

- $\alpha_{i,j}$: amount of load that sink K_j shall request from source S_i in an iteration.
- α_j: fraction of the total load L that sink K_j shall consider in an iteration.
- $B_j^{(q)}$ ($\hat{B}_j^{(q)}$): available (estimated) buffer space in sink K_j in the qth iteration.
- $\hat{B}_j^{(t)}$: time-averaged buffer space at sink K_j, estimated based on historical data.
- L_i: load at source S_i.
- M: total number of sinks in the system, with each sink denoted by K_j, $j = 1, \ldots M$.
- N: total number of sources in the system, with each source denoted by S_i, $i = 1, \ldots N$.
- T: current time in the system.
- $\hat{T}$: estimated processing time for the admitted loads in the system.
- $T^{(q)}$: time taken to process the loads in the qth iteration.
- T_{cp} (T_{cm}): computing (communication) intensity constant.
- T_{di}: deadline requirement of the source S_i.
- T_{ul}: time required to process a unit load.
- w_j: inverse of the computing speed of the sink K_j.
- X_{now} (X_{later}): set of sources that are being processed in an iteration (which shall be processed in a later iteration).
- Y: fraction of the load L that should be taken into consideration in an iteration of installment, where $Y \leq 1$.
- $z_{i,j}$: inverse of the link speed of the link $l_{i,j}$ between source S_i and sink K_j.
- Z_{now}: set of sources that are being considered during the admissibility testing.

4 RADIS Strategies

We now describe our RADIS strategies. In all of our strategies, we assume that the *CN* computes the parameters required by the sink nodes to determine a schedule satisfying the resource constraints.

In the DLT literature [12], it was mentioned that for an optimal scheduling solution, it is necessary and sufficient that all the sinks that participate in the computation stops at the same time instant; else, the loads could be redistributed to improve the processing time. The optimality principle stated in the DLT literature was used, and the load fractions that a sink K_j shall receive from the source S_i was derived in the modified IBS algorithm [10] for systems with prespecified buffer constraints.

The modified IBS algorithm recursively invokes the IBS algorithm [16] and employs a "push-based" strategy. In this scheme, a source node identifies potential sinks (with knowledge about the available resources at the sinks), computes the schedule, and communicates it to other source nodes. Upon receiving this schedule information,

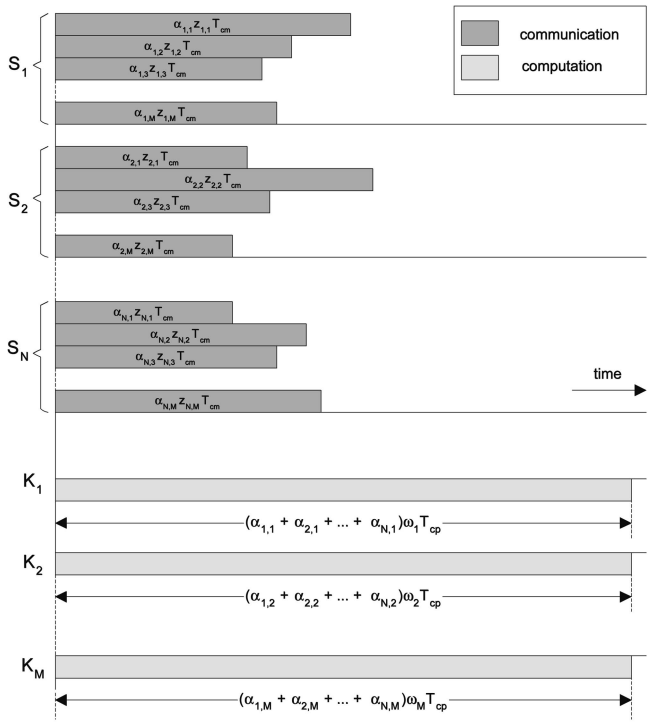

Fig. 2. Timing diagram of the load distribution strategy with N sources and M sinks in an iteration.

all the source nodes send their load portions to the respective sink nodes. Although this algorithm recursively attempts to fill up the buffer space of one or more sinks at every iteration, it is basically an offline algorithm. In this scheme, when a sink's buffer is completely filled up, that sink is not considered for scheduling in the subsequent iterations.

The modified IBS algorithm produces an optimal solution and exhibits finite convergence. However, it does not consider real-life situations, where the buffer capacities at sink nodes vary over time, and the loads to be processed may arrive at arbitrary times to the system.

We consider three different scheduling strategies for such dynamic environments, depending on how the set of loads will be processed. All our strategies work in an incremental fashion, consuming several iterations for scheduling the loads. Each iteration refers to a time period in which a set of sinks is to be scheduled for processing the loads by the CN. Our first strategy utilizes the *Earliest Deadlines First (EDF) Scheme*, wherein among the sources that are admitted into the system, the sources with the earliest deadlines are considered for processing in every iteration. In the second strategy, termed as the *Progressive Scheme*, we process the loads from the sources that are at the risk of missing their deadlines in every iteration. In the third strategy, termed as the *Noninterleaved Scheduling Scheme*, all the sources that were admitted into the system are scheduled for processing in every iteration.

Below, we will describe the workings of our scheduler in the coordinator and sink nodes designed for our strategies in a systematic fashion. The flowcharts shown in Figs. 3, 4, and 5 describe the workings of the CN, the admission

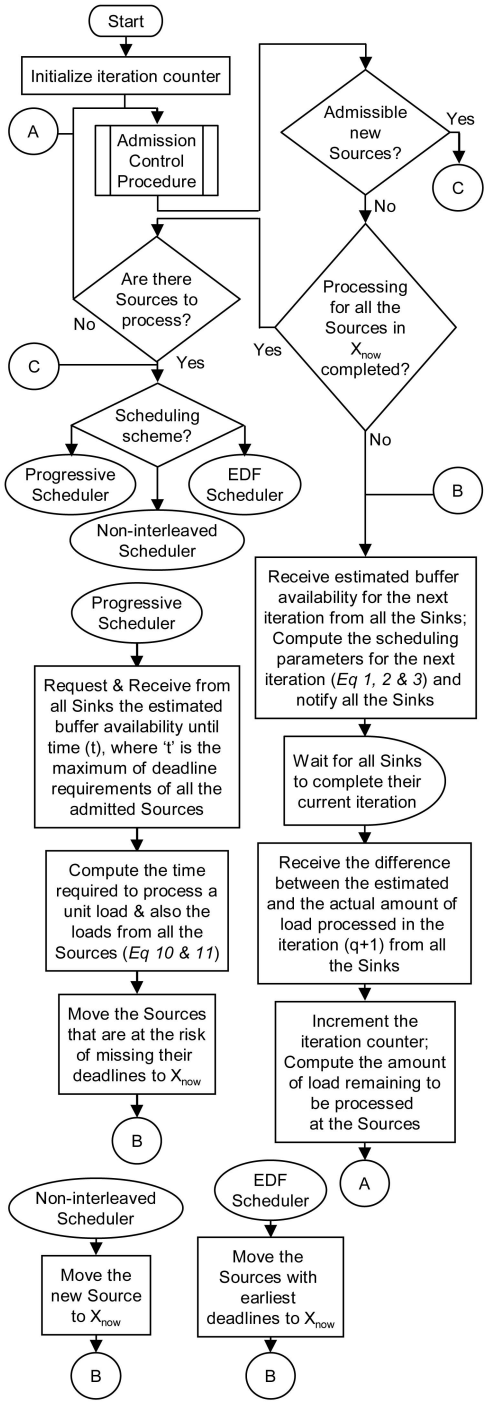

Fig. 3. Workings of the RADIS scheduler at the CN.

control procedure, and the sink nodes, respectively. The pseudocodes for RADIS are made available to the readers on the CS Digital Library (http://www.computer.org/tpds/archives.htm).

In our strategies, in every iteration, after the admissibility testing for the newly arrived sources, the scheduler at the CN first determines the loads to be scheduled and the sinks that will participate. Based on the estimated buffer availabilities at the sink nodes, the CN computes an estimate of the amount of load to be scheduled at a sink node and the finish time for the next iteration. It then

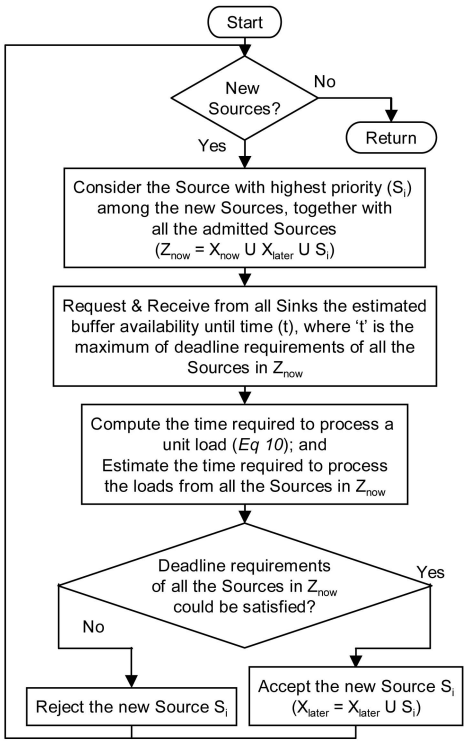

Fig. 4. Admission control procedure at the CN.

broadcasts this schedule information to all the sink nodes. A sink node, upon receiving this information, will wait for the current iteration to be completed and determines the actual buffer availability for the next iteration. Based on the estimate received from the CN and its actual buffer availability, it computes the amount of load that it can process in the next iteration and requests that amount of load from the respective sources. It also communicates the difference between the estimated and the actual amounts of load to the CN.

We shall first consider handling the time-varying buffer availabilities at the sink nodes and then the dynamic arrival of loads. For dynamic environments, a feasible schedule may not exist, unless the sink nodes allow their available buffers to be reused after a given load is processed. Hence, we assume that the sink nodes allow their available buffer spaces to be reused after the processing is completed in an iteration so as to enable the scheduling of more amounts of loads, employ the modified IBS algorithm [10] in every iteration, and incrementally process the loads. Our scheduling strategies take into account that the buffer space variations may not be known a priori. In our strategies, the sinks estimate the amount of buffer space that they could offer for scheduling in the next iteration, as described later in Section 4.1, and communicate it to the CN. With this information, the CN determines the participating sink nodes for the next iteration and computes the required parameters to schedule the loads in an incremental fashion, satisfying the resource constraints as follows.

The timing diagram shown in Fig. 2 represents the communication and computation times of the sources and sinks within a cluster system in an iteration, with the x-axis representing the time. Assuming the buffer availability at a

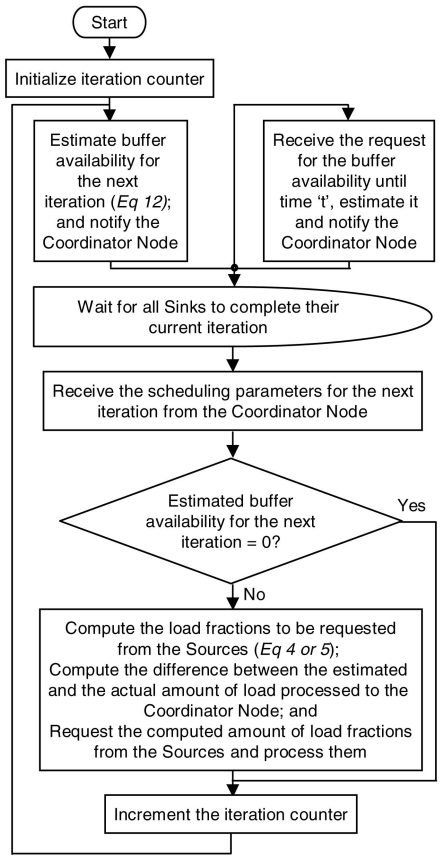

Fig. 5. Workings of the RADIS scheduler at the sink nodes.

sink node in an iteration, say q, as $B_j^{(q)}$ and the load fraction that a sink shall request is proportional to the size of the load at the sources, the fraction of the total load to be taken into consideration in that iteration for optimal processing, denoted as Y, shall be computed as

$$Y = \min\left\{\frac{B_j^{(q)}}{\left(\alpha_j^{(q)} L\right)}, \forall K_j \in P_{now}\right\}, \tag{1}$$

where $Y \leq 1$ and

$$\alpha_j^{(q)} = \frac{1}{w_j\left(\sum_{x=1}^{M} \frac{1}{w_x}\right)}, \forall K_j \in P_{now}. \tag{2}$$

The optimal processing time for all the participating sink nodes at that iteration is given by

$$T^{(q)} = \left(Y \alpha_j^{(q)} L\right) w_j \, T_{cp}. \tag{3}$$

It may be noted that the expression for the load fractions $\left(Y \alpha_j^{(q)} L\right)$ in (3) guarantees that at each iteration, all the participating sink nodes complete their processing at the same time instant. Having thus computed the required parameters for scheduling, the CN communicates them to all the sink nodes.

The sink nodes receive the information from the CN and wait till the processing is completed by all the sink nodes in

the previous iteration. Then, if at a sink, the actual buffer availability is not sufficient to accommodate the estimated amount of load, that sink node computes the load fractions to be requested from the source nodes as

$$\alpha_{i,j}^{(q)} = L_i \cdot \frac{B_j^{(q)}}{L}. \tag{4}$$

If the buffer availability at a sink node is more or enough to accommodate the estimated load fraction, then the sink node computes the load fraction to be requested from the source node S_i in the iteration q as

$$\alpha_{i,j}^{(q)} = Y\alpha_j^{(q)} L_i. \tag{5}$$

The sink nodes communicate to the CN the difference between the amount of load that they estimated to process and the actual amount of load that they are processing. All the sinks request these load fractions from the sources and process them. Following our model described in Section 3, all the sinks start computing the load fractions as they start receiving them from the sources.

The CN receives the information on the difference between the amount of load estimated to be processed and the actual amount of load processed at the participating sink nodes, computes the remaining amount of load in the system, and waits till the processing is completed by all the sink nodes for that iteration. It may be noted that since buffer availability is a function of time, the guarantees given to the sources (to complete the processing within their deadlines) may be met only when the buffer estimation strategy utilized is conservative.

In RADIS, the total load processed in the iteration q is given by

$$\sum_{j=1}^{M} \alpha_j^{(q)} = \sum_{i=1}^{N}\sum_{j=1}^{M} \alpha_{i,j}^{(q)}. \tag{6}$$

Hence, the time taken to process a unit load in that iteration is given by

$$T_{ul} = \frac{T^{(q)}}{\sum_{i=1}^{N}\sum_{j=1}^{M} \alpha_{i,j}^{(q)}}. \tag{7}$$

RADIS attempts to completely fill at least one of the sinks' buffers in every iteration, depending on the processing speed and the size of the buffer available at the sinks. Since, in our strategy, all the sinks are forced to stop processing at the same time, the product of the buffer utilization and the inverse of the computing speed for each sink node will be the same. From this, the maximum buffer utilization or, in other words, the total load that could be processed at each sink in an iteration could be derived as

$$\sum_{i=1}^{N} \alpha_{i,j} = \frac{B_{i^*} w_{i^*}}{w_j}, \text{ where } i^* = argmin\left\{\frac{B_j}{\alpha_j}\right\}. \tag{8}$$

In (8), the "$argmin$" term identifies a sink whose buffer is completely filled. The buffer utilization at other sink nodes and, hence, the total amount of buffer utilized for the optimal processing by the system or, in other words, the total load that could be processed by the system in an iteration could be computed as

$$\sum_{i=1}^{N}\sum_{j=1}^{M} \alpha_{i,j} = \sum_{j=1}^{M} \frac{B_{i^*} w_{i^*}}{w_j}. \tag{9}$$

Substituting (3), (8), and (9) into (7), the time taken to process a unit load is given by

$$T_{ul} = \frac{T_{cp}}{\sum_{j=1}^{M} \left\{\frac{1}{w_j}\right\}}. \tag{10}$$

Hence, the estimated time taken to process the loads in the system is given by

$$\hat{T} = T_{ul} \cdot \sum_{i=1}^{N} L_i. \tag{11}$$

Thus, the time taken to process the loads from the sources that are being considered are estimated in RADIS strategies.

Now, we shall discuss how our strategies consider the dynamic arrival of loads. In RADIS, at the start of every iteration, the CN checks the feasibility for admitting new sources (based on their deadline requirements and priorities), if there are any, and decides on the set of sources to be scheduled in that iteration. The admission criteria for the sources vary for different RADIS schemes, and they are discussed in Section 4.2. The set of loads that are scheduled in an iteration also depends on the chosen scheme of the scheduler: sources with the earliest deadlines are chosen in the case of the EDF Scheduling Scheme, all the sources that are admitted into the system are chosen in the case of the Noninterleaved Scheduling Scheme, and the set of sources that needs to be scheduled so as to meet the deadline requirements is chosen in the case of the Progressive Scheduling Scheme.

While determining the set of sources to be scheduled in the case of the Progressive Scheme, there are three possibilities, and the actions taken under such situations are as follows:

1. *The deadline requirements for all the sources considered are later than* $\hat{T}$. Under this condition, the deadline requirements of all the sources in the system could be satisfied. Hence, the set of sources that were considered previously is scheduled.
2. *The deadline requirements for all the sources considered are earlier than* $\hat{T}$. Under this condition, there is a chance that the deadline requirements of a set of sources that were considered previously may not be satisfied. Hence, those sources are scheduled immediately.
3. *The deadline requirements for some of the sources are earlier, and some are later than* $\hat{T}$. Under this condition, we reiterate considering those sources whose deadlines are earlier than $\hat{T}$.

The new set of loads and the unprocessed loads from the existing sources are considered together for scheduling at the start of every iteration. This process is continued until all the loads are processed.

4.1 Buffer Estimation Strategy

We propose a distributed buffer estimation strategy based on the weighted average calculations of the buffer availability in the previous "s" iterations. The weights for computing the estimates are based on the iteration indices until the current iteration. Our estimation algorithm shall be executed at all sink nodes. A sink node, after estimating the buffer space to render in the next iteration, shall communicate it to the *CN* so that it could determine the scheduling parameters required for the sink nodes.

For estimating the buffer availability in a sink, each sink K_j needs to keep track of the actual buffer sizes B_j from its previous "s" iterations. In an iteration q, each sink node shall estimate the buffer size that will be available for the next iteration $(q+1)$ as

$$\hat{B}_j^{(q+1)} = \left(\frac{\sum_{k=0}^{(s-1)} \left((s-k) \cdot B_j^{(q-k)} \right)}{\sum_{k=0}^{(s-1)} k} \right) \cdot p, \qquad (12)$$

and declare it to the CN. In (12), p is the probability that the estimated buffer size will be available at a sink at the next iteration. The value of p can be chosen based on the confidence level of the buffer estimator. For practical purposes, we shall assume that p equals 0.95. This guarantees that the expected buffer sizes will be available at the sinks, with a confidence level of 95 percent, for the next iteration.

4.2 Admission Control Strategy

In RADIS, in every iteration, if there are new sources, then the CN considers them in their priority order. It then requests the sink nodes to estimate their buffer availabilities until the farthest deadline requirement time of all the sources that were accepted earlier and also the new source that is being considered. The sinks estimate their buffer availability by calculating the time average of their historical data as

$$\hat{B}_j^t = \frac{1}{t_{req}} \int_{t=(T-t_{req})}^{t=T} B_j(t)dt, \qquad (13)$$

where $t_{req} = (\max\{T_{di}\} - T), (T - t_{req}) \geq 0$, and $\max\{T_{di}\}$ is the farthest deadline requirement time of all the sources.

The estimation requests could be further minimized if the CN requests the sink nodes to estimate the buffer variations, taking into consideration the deadline requirements of all the new sources in every iteration. This approach has an inherent advantage of minimizing the communication required for estimating the buffer for admissibility testing, especially when the source arrival rates are higher.

In the case of the EDF Scheme, the algorithm checks the deadline requirement of all the sources that were admitted earlier and also the new source against the processing time required to process them, considering the sources with the earliest deadlines. The process is repeated until the deadline requirements of all of the sources are found to be satisfied, in which case the new source shall be admitted, or the deadline requirement of some of the sources is violated, in which case the new source shall not be admitted into the system.

In the case of the Progressive Scheme, during the admissibility testing for a new source, there are three possibilities, and different actions are taken under such situations. They are as follows:

1. *The deadline requirements for all the sources considered are later than* $\hat{T}$. Under this condition, the new source that is considered shall be admitted.
2. *The deadline requirements for all the sources considered are earlier than* $\hat{T}$. Under this condition, the new source that is considered shall not be admitted. Note that the priority of a source could depend on its processing requirements.
3. *The deadline requirements for some of the sources are earlier and some are later than* $\hat{T}$. Under this condition, we reiterate considering only those sources whose deadlines are earlier than $\hat{T}$.

In the case of the Noninterleaved Scheduling Scheme, if the deadline requirement of all the sources that were admitted earlier and also the new source could be satisfied when they all are scheduled together in every iteration, then the new source shall be admitted; otherwise, the new source shall not be admitted into the system.

The proposed admissibility criteria, together with the conservative buffer estimation strategy, guarantees that the deadlines for all the loads that are accepted will always be satisfied. It may be noted that in RADIS, the priorities of the sources are used only to resolve conflicts that may arise while admitting multiple sources. If multiple sources have identical priorities set, then schemes such as first in, first out (FIFO) or other possible heuristics can be adopted to resolve the conflict.

The interleaving scheduling strategy of RADIS works in a style contrary to most of the conventional schedulers, which use priority as the criterion for scheduling the loads. In this scheme, some of the sources that are processed in an iteration could be suspended, and some other sources could be scheduled in the following iteration so as to satisfy the deadline requirements of the admitted sources. A simulation study presented in Section 5 clarifies the workings of RADIS and all the schemes mentioned above in detail.

5 Performance Evaluation

In this section, we shall describe our experimental platform and list certain parameters that influence the performance of the methodologies. As a grid computing environment is envisaged as a large-scale system, in our study, we simulated 64, 128, and 256-node (sink) systems. The computing intensity constant (T_{cp}) and sink speed $\left(\frac{1}{w_j}\right)$ parameters are derived from the Solenoidal Tracker at Relativistic Heavy-Ion Collider experiments conducted at Brookhaven National Laboratory [10]. The speed parameter is assigned to the sink nodes based on the uniform probability distribution.

In our study, the buffer availabilities at the sinks are allowed to vary randomly over time (referred to as the *Time Varying Buffer* (*TVB*) scenario), and we also observe the best case performance when the buffer sizes are time invariant

TABLE 1
Simulation Parameters and Their Range of Values

Parameter	Range of values
Simulation period [in seconds]	50,000
Load arrival rates [in arrivals / second]	$0.001 - 1.0$
Number of Sinks	$64,\ 128,$ and 256
Window size for Buffer Estimation Strategy (s)	8
Inverse of Sink Speeds (w_j)	1.11×10^{-9}, 6.25×10^{-10}, 5.00×10^{-10} and 3.57×10^{-10}
Computing Intensity Constant (T_{CP})	6.52×10^{10}
Load Sizes (Type I \ Type II)	$(50 - 70)\backslash(170 - 190)$
Load Deadlines (Type I \ Type II) [in seconds]	$(500 - 750)\backslash(6000 - 7000)$
Priority of Sources	$0 - 10$
Buffer Sizes (Small \ Medium \ Large)	$(0,\ 0.5)\backslash(0,\ 0.75)\backslash(0,\ 0.5,\ 1.0)$
Buffer Size Variations (Slow \ Medium \ Fast)	$(3000 - 3500)\backslash(2000 - 2500)\backslash$ $(1000 - 1500)$

(referred to as the *Time Invariant Buffer* (TIB) scenario). The TIB scenario results are important, as they provide the upper bounds for the performance of the strategies. Also, all possible variations (for the priority of loads and available buffer sizes), ranging from small to large fluctuations, and the frequency of the buffer availability fluctuations are captured. These are varied randomly by following the uniform probability distributions, whereas the load arrival rates follow the Poisson distribution. In our simulations, we consider three different sets of loads. Set 1 consists of 10 percent type-I and 90 percent type-II load sizes (see Table 1). Set 2 consists of 50 percent type-I and 50 percent type-II load sizes. Set 3 consists of 90 percent type-I and 10 percent type-II load sizes. All the above-mentioned simulation parameters and their respective ranges are given in Table 1.

In our experiments, the number of sinks participating in every iteration is also allowed to vary, simulating the nodes leaving the system or participating in the computation in a random fashion. Thus, we attempt to capture the behavior of the strategies close to a real-life environment. Further, our schemes guarantee that the deadlines for all the loads that are accepted will be satisfied, since in our experiments, the buffer sizes follow the uniform probability distribution, and our schemes utilize the time-averaged buffer estimation strategy for admissibility testing.

5.1 Metrics of Interest

We consider the following performance metrics, which are of direct relevance to this study. The number of loads accepted in the TVB scenario (Φ) has a higher significance, since one of our aims is to maximize the number of loads that are accepted. We also define the acceptance ratio (β) and the ratio of acceptance ratios (η) as

$$\begin{aligned} \beta &= \frac{Number\ of\ loads\ accepted}{Number\ of\ loads\ arrived}, \\ \eta &= \frac{\beta_{TVB}}{\beta_{TIB}}, \end{aligned} \tag{14}$$

where β_{TVB} and β_{TIB} are the acceptance ratios of the TVB and TIB scenarios, respectively.

Second, for the TVB scenario, we define a metric (γ) that quantifies the throughput of the system as

$$\gamma = \frac{Number\ of\ loads\ processed}{Number\ of\ loads\ accepted}. \tag{15}$$

Finally, we define the average buffer utilization in an iteration at a sink node (ζ) and the ratio of the average buffer utilization (χ) as

$$\begin{aligned} \zeta &= \frac{\sum_{j=1}^{M} \left(\left(\sum_{i=1}^{q} \frac{\min\{Y\alpha_j^{(i)} L, B_j^{(i)}\}}{B_j^{(i)}} \right) \Big/ q \right)}{M}, \\ \chi &= \frac{\zeta_{TVB}}{\zeta_{TIB}}, \end{aligned} \tag{16}$$

where q is the number of iterations that the sink node has participated, and ζ_{TVB} and ζ_{TIB} are the average buffer utilization in an iteration at a sink node in the TVB and TIB scenarios, respectively.

5.2 Discussion of the Results

Fig. 6 shows the behavior of Φ, β_{TVB}, η, γ, and χ when we employ RADIS in a system with 64 sinks with respect to the load arrival rates for two different deadline types, as given in Table 1. Though we simulated 64, 128, and 256-node (sink) systems, since all our strategies exhibited identical behavior in terms of trends for all the performance metric considered, we have presented here only the results for the 64-node system. In Fig. 6, we denote the Progressive Scheme with load sets 1, 2, and 3 as $PS1$, $PS2$, and $PS3$, respectively, the EDF Scheme with load sets 1, 2, and 3 as $ES1$, $ES2$, and $ES3$, respectively, and the Noninterleaved Scheduling Scheme with load sets 1, 2, and 3 as $NS1$, $NS2$, and $NS3$, respectively.

In Fig. 6, it is observed that at low arrival rates (less than 0.006), there is little difference in Φ for the various scheduling schemes. As the arrival rate increases, irrespective of the scheduling schemes and the deadline type of the loads, the number of loads accepted for load set 3 is higher than that for load set 2, which, in turn, is higher than that for load set 1. It is also interesting to observe that a better performance is shown by the Progressive Scheduling Scheme in the case of loads with type-I deadlines, and by the EDF Scheduling Scheme for the loads with type-II deadlines. Also, the performance improvement at higher arrival rates for loads with type-II deadlines is significantly higher for the EDF scheme when compared with other schemes. However, the improvement is of less significance for arrival rates higher than 0.3, since the β_{TVB} is closer to 0 at these rates (see the plot of β_{TVB} in Fig. 6).

Initially, all the arriving loads get accepted by the system (the acceptance ratio is close to 1), and as the arrival rate increases further, it starts falling steeply (the zone represented as "A" in the plot of β_{TVB} in Fig. 6). This is due to the fact that the scheduler can no longer continue to accept the

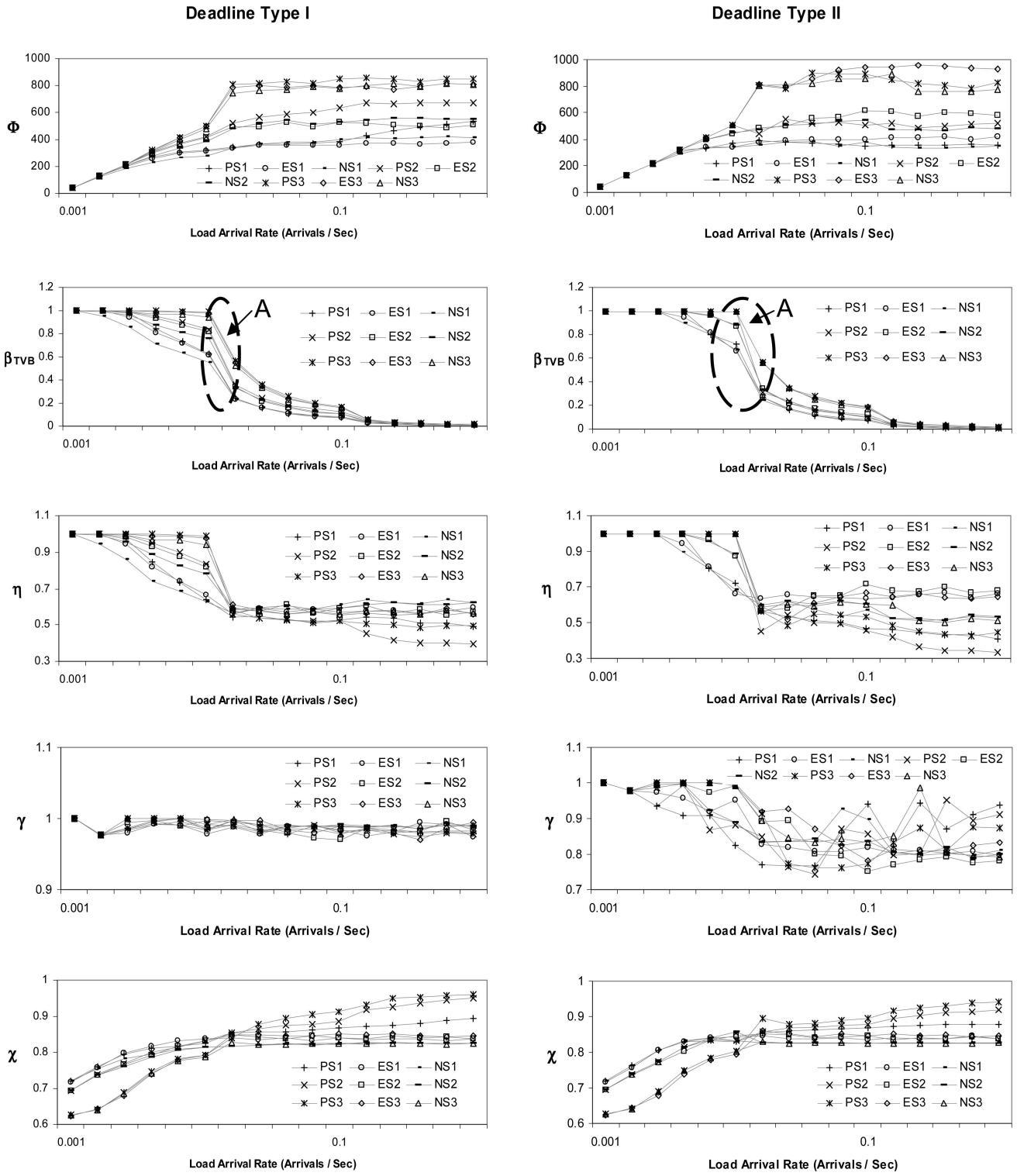

Fig. 6. Simulation results for the Progressive, EDF, and Noninterleaved Scheduling RADIS schemes for load sets 1, 2, and 3 and deadline types I and II in a 64-node system.

newly arrived loads, unless the deadline requirements of the already accepted loads, together with the new load being considered, could be satisfied. Hence, the admissibility testing starts rejecting some of the newly arrived loads. As the arrival rate further increases, the acceptance ratio β_{TVB} moves closer to 0.

Based on the plot of η in Fig. 6, it is observed that at arrival rates lower than 0.03, for all load sets and deadline types, η is almost similar for both the EDF and Progressive Schemes of RADIS because the acceptance ratios (β) of these schemes are almost identical. At these arrival rates, for loads with type-I deadlines, these η values are higher than

that for the Noninterleaved Scheduling Scheme. At higher arrival rates, for all load sets with type-I deadlines, η tends to saturate close to a value of 0.6 for the EDF and Noninterleaved schemes and about 0.4-0.5 for the Progressive Scheme. For all load sets with type-II deadlines, η tends to saturate close to a value of 0.65 for the EDF Scheme, 0.5 for the Noninterleaved Scheme, and around 0.3-0.4 for the Progressive Scheme.

It is also observed that the system throughput γ value is closer to 1 for loads with type-I deadline requirements, irrespective of the load arrival rates, whereas it decreases with increasing arrival rates for loads with type-II deadline requirements for all the load sets and schemes. In the case of the EDF Scheme, though the system throughput decreases with the increase in load arrival rates for loads with type-II deadlines, it is seen to be more robust, since the variations in the system throughput are less as compared with other schemes (refer to the plot of γ in Fig. 6).

The plot of χ in Fig. 6 shows that at arrival rates lower than 0.03, for both type-I and type-II deadline requirements of the loads, the average buffer utilization for all the schemes are almost identical, the utilization of load set 1 is higher than that of load set 2, and the utilization of load set 2 is higher than that of load set 3. For arrival rates higher than 0.03, for both deadline types and all the load sets, the utilization saturates at a value of around 0.8 in the case of both the EDF and Noninterleaved Scheduling Schemes. In the case of the Progressive Scheme, the trend reverses, then, the utilization of load set 3 becomes higher than that of load set 2, and the utilization of load set 2 becomes higher than that of load set 1, and the values saturate between 0.85 and 0.95 at higher arrival rates.

It is to be noted that at arrival rates lower than 0.006, the number of loads accepted for all the schemes are almost the same, and at higher arrival rates, although the acceptance ratios of all the schemes are almost similar, for loads with type-I deadlines, the average buffer utilization is higher, and the number of loads that are accepted are also higher in the case of the Progressive Scheme, whereas for loads with type-II deadlines, the average buffer utilization is lower, and the number of loads that are accepted are higher in the case of the EDF Scheme. The implementation of the Noninterleaved Scheduler is simpler than other schemes.

In an actual system, we propose to have a decision-making mechanism that monitors the load arrival patterns and their deadline requirement at the *CN* and dynamically choose the appropriate scheduling scheme. Hence, irrespective of the type of loads and their deadline requirements, when the load arrival rate is lower, we propose to utilize the Noninterleaved Scheduling Scheme. When the load arrival rate increases further, depending upon the type of deadline requirements of the loads, we propose to utilize either the Progressive or the EDF Scheduling Schemes. However, when the load arrival rate is higher than 0.3, we propose to utilize the simpler Noninterleaved Scheduling Scheme, since the acceptance ratio (β_{TVB}) is closer to 0. Thus, all the proposed schemes are very useful for real-life systems.

6 Conclusions

In this paper, we have proposed distributed scheduling strategies for processing multiple divisible loads on grid systems. As in real-life situations, we have considered the dynamic arrival of loads, the buffer capacity constraints at the sink nodes, and the deadline requirement of the loads to be processed. We have proposed three schemes of RADIS and have rigorously analyzed and evaluated their performance. In our simulations, for all the schemes, the number of sinks that participate in the processing in an iteration are allowed to vary, which is reflective of real-life situations. The impact of load sizes, load deadlines, and buffer size variations are also captured in our simulation study.

In the schemes proposed in this paper, the CN performs the admissibility test, determines the loads to be processed and the sinks that participate in an iteration, computes the scheduling parameters required for the sink nodes for determining the schedule, and communicates them to the sink nodes. The sink nodes perform the buffer estimation, estimate the amount of load fractions to be processed based on the scheduling parameters received from the CN, determine the amount of load fractions to be requested from the source nodes based on the actual buffer availability at the start of an iteration, and communicate the difference between the estimated and the actual amounts of load processed in an iteration to the CN. The sink nodes also request and process the amount of load fractions thus determined. In our schemes, the scheduling is done in a distributed manner (the load fractions are computed at the sink nodes), and only $(4 + r)$ variables (where r is the number of sources that are scheduled in an iteration) are communicated from the CN to the sink nodes. Hence, the effectiveness of our schemes is more pronounced in larger systems.

The strategies shown here are explicitly designed for a group of computing nodes within a cluster node, which is part of a grid infrastructure. The proposed strategies can be tuned to work across several cluster nodes by considering communication delays. One of the ways on how the strategies could be tuned follows the work presented in [6] for Ethernet networks. Another limitation in our schemes is the single-point admissibility testing performed by the CN. However, one could implement a distributed approach by using leader-election-like algorithms [20] to make it more fault tolerant.

We infer and observe the following based on our work presented in this paper:

- The DLT paradigm has been proven to be a valid tool for handling large-scale computational loads on cluster/grid systems.
- Though our algorithm is shown to provide the best effort schedules, the underestimation of buffer availability enables our scheme not to miss the deadlines for the accepted loads.
- All the proposed scheduling strategies are scalable, are relevant in real-life situations, and are shown to be useful under different scenarios.
- In grid systems, with high-speed links, concurrent communication in different links is certainly a viable

model for handling large-scale computational loads such as the one addressed in this paper. However, when the underlying grid system has slower links, for handling multiple loads, alternative models with nonzero communication delays, which are proposed using the DLT paradigm [1], should be considered.

- Although we have proposed a scheme for buffer estimation at sink nodes, it may be noted that the design of the buffer estimation strategy is a topic in itself, and other strategies such as the use of a fading memory could be "plugged into" the RADIS strategies.

ACKNOWLEDGMENTS

Bharadwaj Veeravalli's research was funded by the National Grid Office, Singapore, under the grant R-263-000-350-592. Sivakumar Viswanathan's research was supported by the Institute for Infocomm Research, Singapore. Thomas G. Robertazzi would like to acknowledge the useful conversations with Dantong Yu.

REFERENCES

[1] H.M. Wong, V. Bharadwaj, and B. Gerassimos, "Design and Performance Evaluation of Load Distribution Strategies for Multiple Divisible Loads on Heterogeneous Linear Daisy Chain Networks," *J. Parallel and Distributed Computing,* vol. 65, no. 12, pp. 1558-1577, Dec. 2005.

[2] D. Ghose, H.J. Kim, and T.H. Kim, "Adaptive Divisible Load Scheduling Strategies for Workstation Clusters with Unknown Network Resources," *IEEE Trans. Parallel and Distributed Systems,* vol. 16, no. 10, pp. 897-907, Oct. 2005.

[3] L. Ping, B. Veeravalli, and A.A. Kassim, "Design and Implementation of Parallel Video Encoding Strategies Using Divisible Load Analysis," *IEEE Trans. Circuits and Systems for Video Technology,* vol. 15, no. 9, pp. 1098-1112, Sept. 2005.

[4] L. Marchal, Y. Yang, H. Casanova, and Y. Robert, "A Realistic Network/Application Model for Scheduling Divisible Loads on Large-Scale Platforms," *Proc. 19th Int'l Parallel and Distributed Processing Symp. (IPDPS '05),* p. 48b, Apr. 2005.

[5] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, and Y. Yang, "Scheduling Divisible Loads on Star and Tree Networks: Results and Open Problems," *IEEE Trans. Parallel and Distributed Systems,* vol. 16, no. 3, pp. 207-218, Mar. 2005.

[6] B. Veeravalli, "Design and Performance Analysis of Heuristic Load Balancing Strategies for Processing Divisible Loads on Ethernet Clusters," *Int'l J. Computers and Applications,* vol. 27, no. 2, pp. 97-107, 2005.

[7] M.A. Moges, D. Yu, and T.G. Robertazzi, "Grid Scheduling Divisible Loads from Multiple Sources via Linear Programming," *Proc. 17th Int'l Conf. Parallel and Distributed Computing and Systems (PDCS '04),* pp. 423-428, Nov. 2004.

[8] M. Wu and X.H. Sun, "Memory-Conscious Task Partition and Scheduling in Grid Environments," *Proc. Fifth IEEE/ACM Int'l Workshop Grid Computing (Grid '04),* pp. 138-145, Nov. 2004.

[9] S. Kim and J.B. Weissman, "A Genetic Algorithm-Based Approach for Scheduling Decomposable Data Grid Applications," *Proc. 33rd Int'l Conf. Parallel Processing (ICPP '04),* vol. 1, pp. 406-413, Aug. 2004.

[10] H.M. Wong, D. Yu, B. Veeravalli, and T.G. Robertazzi, "Data-Intensive Grid Scheduling: Multiple Sources with Capacity Constraints," *Proc. 16th Int'l Conf. Parallel and Distributed Computing and Systems (PDCS '03),* pp. 7-11, Nov. 2003.

[11] A.E. Darling, L. Carey, and W. Feng, "The Design, Implementation and Evaluation of mpiBLAST," *Proc. Fourth LCI Int'l Conf. Linux Clusters: The HPC Revolution 2003,* June 2003.

[12] V. Bharadwaj, D. Ghose, and T.G. Robertazzi, "Divisible Load Theory: A New Paradigm for Load Scheduling in Distributed Systems," *Cluster Computing on Divisible Load Scheduling,* vol. 6, no. 1, pp. 7-18, Jan. 2003.

[13] H.J. Kim, "A Novel Optimal Load Distribution Algorithm for Divisible Loads," *Cluster Computing on Divisible Load Scheduling,* vol. 6, no. 1, pp. 41-46, Jan. 2003.

[14] B. Veeravalli and S. Ranganath, "Theoretical and Experimental Study on Large-Size Image Processing Applications Using Divisible Load Paradigm on Distributed Bus Networks," *Image and Vision Computing,* vol. 20, nos. 13-14, pp. 917-936, Dec. 2002.

[15] S.K. Chan, V. Bharadwaj, and D. Ghose, "Large Matrix-Vector Products on Distributed Bus Networks with Communication Delays Using the Divisible Load Paradigm: Performance Analysis and Simulation," *Math. and Computers in Simulation,* vol. 58, pp. 71-79, 2001.

[16] X. Li, V. Bharadwaj, and C.C. Ko, "Divisible Load Scheduling on Single-Level Tree Networks with Buffer Constraints," *IEEE Trans. Aerospace and Electronic Systems,* vol. 36, no. 4, pp. 1298-1308, Oct. 2000.

[17] M. Drozdowski and P. Wolniewicz, "Experiments with Scheduling Divisible Tasks in Clusters of Workstations," *Proc. Sixth European Conf. Parallel Computing (Euro-Par '00),* pp. 311-319, 2000.

[18] J. Blazewicz, K. Ecker, B. Plateau, and D. Trystram, *Handbook on Parallel and Distributed Processing.* Springer, 2000.

[19] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, 1999.

[20] H. Attiya and J. Welch, *Distributed Computing: Fundamentals, Simulations and Advanced Topics.* McGraw Hill, 1998.

[21] D.A.L. Piriyakumar and C.S.R. Murthy, "Distributed Computation for a Hypercube Network of Sensor-Driven Processors with Communication Delays Including Setup Time," *IEEE Trans. Systems, Man, and Cybernetics, Part A,* vol. 28, no. 2, pp. 245-251, Mar. 1998.

[22] V. Bharadwaj, D. Ghose, V. Mani, and T.G. Robertazzi, *Scheduling Divisible Loads in Parallel and Distributed Systems.* CS Press, Sept. 1996.

[23] R. Agrawal and H.V. Jagadish, "Partitioning Technologies for Large-Grained Parallelism," *IEEE Trans. Computers,* vol. 37, no. 12, pp. 1627-1634, Dec. 1988.

[24] Y.C. Cheng and T.G. Robertazzi, "Distributed Computation with Communication Delays," *IEEE Trans. Aerospace and Electronic Systems,* vol. 24, no. 6, pp. 700-712, Nov. 1988.

Sivakumar Viswanathan received the BE degree from the University of Madras, India, in 1989 and the MSc degree from the National University of Singapore (NUS), Singapore, in 2001. He is currently pursuing the PhD degree in the area of high-performance network-based computing at NUS and is the assistant department manager in the Embedded Systems Department, Institute of Infocomm Research, Singapore. His research interests include multiprocessor systems, cluster/grid computing, and scheduling in parallel and distributed systems. He is a student member of IEEE.

Bharadwaj Veeravalli received the BSc degree in physics from Madurai-Kamaraj University, India, in 1987, the master's degree in electrical and communications engineering from the Indian Institute of Science (IISc), Bangalore, India, in 1991, and the PhD degree from the Department of Aerospace Engineering, IISc, in 1994. He did his postdoctoral research in the Department of Computer Science, Concordia University, Montreal, in 1996. He is currently an associate professor in the Department of Electrical and Computer Engineering, Communications and Information Engineering Division, National University of Singapore, Singapore. He is also currently serving the editorial board of the *IEEE Transactions on Computers*, the *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, and the *International Journal of Computers and Applications* as an associate editor. His mainstream research interests include multiprocessor systems, cluster/grid computing, scheduling in parallel and distributed systems, bioinformatics, and multimedia computing. He has published extensively in high-quality international journals and conferences and has coauthored three research monographs in the areas of parallel and distributed systems, distributed databases, and networked multimedia systems. He is a senior member of the IEEE and the IEEE Computer Society.

Thomas G. Robertazzi received the BEE degree from the Cooper Union, New York, in 1977 and the PhD degree from Princeton University, Princeton, New Jersey, in 1981. He is currently a professor in the Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, New York. In supervising a very active research area, he has published extensively in the areas of parallel processor and grid scheduling, ad hoc radio networks, telecommunications network planning, asynchronous transfer mode (ATM) switching, queuing, and Petri networks. He has authored, coauthored, or edited four books in the areas of performance evaluation, scheduling, and network planning. He is a fellow of the IEEE.