Design and Analysis of a Dynamic Scheduling Strategy with Resource Estimation for Large-Scale Grid Systems

Sivakumar Viswanathan, Bharadwaj Veeravalli
Department of Electrical and Computer Engineering, National University of Singapore
4 Engineering Drive 3, Singapore 117576
{g0306272, elebv}@nus.edu.sg

Dantong Yu
Department of Physics, Brookhaven National Laboratory, Upton, NY 11973, USA
dtyu@bnl.gov

Thomas G. Robertazzi
Department of Electrical and Computer Engineering, Stony Brook University
Stony Brook, NY 11794, USA
tom@ece.sunysb.edu

Abstract

In this paper, we present a resource conscious dynamic scheduling strategy for handling large volume computationally intensive loads in a Grid system involving multiple sources and sinks / processing nodes. We consider a "pull-based" strategy, wherein the processing nodes request load from the sources. We employ the Incremental Balancing Strategy (IBS) algorithm proposed in the literature and propose a buffer estimation strategy to derive optimal load distribution. Here, we consider non-time critical loads that arrive at arbitrary times with time varying buffer availability at sinks and utilize buffer reclamation techniques so as to schedule the loads. We demonstrate detailed workings of the proposed algorithm with illustrative examples using real-life parameters derived from STAR experiments in BNL for scheduling large volume loads.

1. Introduction

Large volume computational data that are being generated in the high energy and nuclear physics experiments demand new strategies for collecting, sharing, transferring and analyzing the data. For example, the Solenoidal Tracker At RHIC (STAR) experiment at Brookhaven National Laboratories (BNL) is collecting data at the rate of over a Tera-Bytes/day. The STAR collaboration is a large international collaboration of about 400 high energy and nuclear physicists located at 40 institutions in the United States, France, Russia, Germany, Israel, Poland, and so on. After the Relativistic Heavy-Ion Collider (RHIC) experiments at BNL came on-line in 1999, STAR began data taking and concurrent data analysis that will last about ten years. STAR needs to perform data acquisition and analyzes over approximately 250 tera-bytes of raw data, 1 peta-bytes of derived and reconstructed data per year. The volume of data is expected to increase by a factor of 10 in the next five years. Details on data acquisition and hardware can be found in [12]. These experiments require effective analysis of large amounts of data by widely distributed researchers who must work closely together. Expanding collaborations and intensive data analysis coupled with increasing computational and networking capabilities stimulates a new era of service oriented computing: Grid computing [1].

Grid computing consists of large sets of diverse, geographically distributed resources that are collected into a virtual computer for high performance computation. Grid computing creates middleware and standards to function between computers and networks to allow full resource sharing among individuals, research institutes, and corporate organizations and to dynamically allocate the idle computing capability to the needed users at remote sites. The diversity of these computing resources and their large number of users creates a challenge to efficiently schedule and utilize these resources.

Scheduling is a significant problem in fairly allocating resources in cluster and grid systems. In divisible load scheduling theory (DLT) domain, scheduling loads under time varying processor and link speeds have been studied in [6]. However, to date there has been no work on dynamically scheduling divisible loads (large volume loads) on a Grid environment when the resource availability at sinks varies randomly over time. The motivation for this paper stems from the challenges in managing and utilizing computing resources in Grids as efficiently as possible, with performance optimization as the main focus. The performance metric of interest includes, job throughput, resource utilization, job response time and its variance. We propose an efficient dynamic scheduling strategy for situations where in the jobs needs to be completed as early as possible. We provide detailed analysis of the algorithm with respect to the above metrics and demonstrate the performance using illustrative examples with real-life parameters derived from STAR experiments in BNL. The analytical flexibility offered by DLT is thoroughly exploited to design resource aware algorithms that make best use of the available resources on the grid. It also offers an exciting opportunity to optimally schedule multiple divisible loads in Grid computing.

Our contributions in this paper are multi-fold. We consider the problem of scheduling several loads generated in a Grid system onto a set of nodes, using the DLT paradigm. We design and propose a dynamic scheduling algorithm that considers system level constraints such as finite buffer capacity constraints at the processing nodes. We propose resource reclaiming strategies in our design of the algorithm. The workings of our algorithm is elaborated using numerical example with real-life parameters and data acquired from STAR experiments, described above. Our algorithm is designed to adapt to network and load size scalability. Our study and systematic design clearly elicits the advantages offered by our strategy.

The paper is organized as follows: In section 2 we provide the research background and related work for Grid scheduling and DLT. In section 3 we formalize the multi-source and multi-sink problem in a Grid system. In section 4 we discuss the load distribution strategy and provide an incremental scheduling strategy for the dynamic environment. In section 6 we discuss the scheduling algorithm and highlight its advantages and provide the conclusion.

2. Related Work

In this section, we shall now present some of the related work relevant to the problem addressed in this paper. For divisible loads, research since 1988 has established that optimal allocation/scheduling of divisible load to processors and links can be solved through the use of a very tractable linear model formulation, referred to as DLT. It is rich in such features as easy computation, a schematic language, equivalent network element modelling, results for infinite sized networks and numerous applications. This linear theory formulation opens up striking modelling possibilities for systems incorporating communication and computation issues, as in parallel, distributed and Grid computing. Optimality here 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 solution or, in simpler models, through recursive algebra. 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, such as store and forward and virtual cut through switching and the presence or absence of front end processors. Front end processors allow a processor to both communicate and compute simultaneously by assuming communication duties [8]. There exists literature of some sixty journal papers on DLT. In addition to the monograph [8], two introductory up-to-date surveys have been published recently [4, 9]. The DLT theory has been proven to be remarkably flexible in the sense that the model allows analytical tractability to derive a rich set of results regarding several important properties of the proposed strategies and to analyze their performance. Consequently, we do not attempt to present another survey here; however, we refer to the following papers that are either directly or indirectly relevant to the context of this paper. A very directly relevant material to the problem addressed in this paper is [3] in which authors propose an Incremental Balancing Strategy (IBS) to accommodate divisible loads when there are buffer constraints in the processors. An alternative scheme for dynamic environments that considers admission control mechanisms has been studied recently in [10]. Issues such as processor release times coupled with buffer capacity constraints are studied in [7]. The solution time (time at which the processed loads/solution is made known at the originator) is discussed in [5]. We refer astute readers to the recent surveys mentioned above for an up-to-date literature in this domain.

3. Problem Formulation and Some Remarks

The Grid computing system to be considered here comprises of N control processors, referred to as *sources*, that have load to be processed and M computing elements, referred to as *sinks*, for processing loads, as shown in Figure 1. Each sink might include one supercomputer or a cluster of computers connected by local area networks and controlled by a header (root) node, and may have different computing and communication capabilities. In a simplified view, these clusters of processors can be replaced with a single equivalent processor. The grid computing system can then be modelled as a fully connected bi-partite graph as in Figure 2: a set of graph vertices could be decomposed into two disjoint

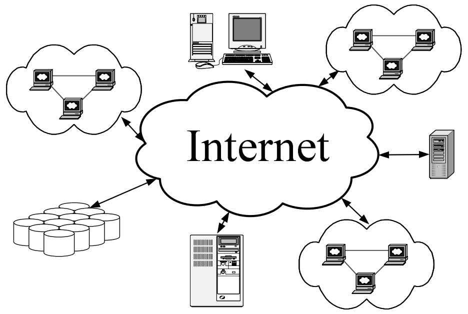

Figure 1. Grid System

sets such that no two graph vertices within the same set are adjacent, while any pair of two graph vertices from these two sets are adjacent. Thus this bi-partite graph is a representation of the fact that each source can schedule its load on all the sinks.

In real-life situations, one of the practical constraints is the availability of the resources on a Grid. For instance, this resource could be the memory capacity that a sink can offer to the sources. Thus, whenever two or more sources compete for a sink, the available memory (equivalently, the amount of load that can be accepted for processing from each source) is to be shared among such sources. We precisely consider this real-life constraint in our proposed algorithm. Further, on such a Grid environment, it is possible that one can either follow a *push*-based approach or a *pull*-based approach to distribute and schedule the loads. In a push-based approach, the sources themselves identify potential sinks (with an assumption/knowledge about the available resources at the sinks) and schedule their loads. Whereas, in a pull-based approach, the sinks collect the requests from the competing sources and schedule them depending on the availability of the resources among themselves. Both the schemes have their merits and the choice of the approach depends purely on the application requirements and implementation constraints such as the size of the grid, the resource availability, etc. In this paper, we shall consider a pull-based approach in the design of our scheduling strategy.

Now, we will formally define the problem we address. As described above, we consider a Grid system with N sources denoted as $S_1, S_2, ..., S_N$ and M sinks denoted as $K_1, K_2, ..., K_M$. For each source, there is a direct link to all the sinks and we denote the link between S_i and K_j as $l_{i,j},\ i = 1, .., N,\ j = 1, .., M$. Each source S_i has a load, denoted by L_i to process. Without loss of generality, we assume that all sources can send their loads to all the sinks simultaneously. Similarly, we also assume that all the sinks can request and receive load portions from all sources.

The objective in this study is to schedule all the N loads among M sink nodes such that the *processing time*, defined as the time instant when all the M sinks complete processing the loads, is a minimum. The scheduling strategy is such that the scheduler (without loss of generality we assume that the scheduler resides in K_1) will first obtain the information about the available memory capacities at other sinks, their computing speeds, and the size of the loads from the sources. The scheduler will then calculate and notify each sink on the optimum load fractions that are to be requested from each source. This information can be easily communicated using any of the standard or customized communication protocols without incurring any significant communication overhead.

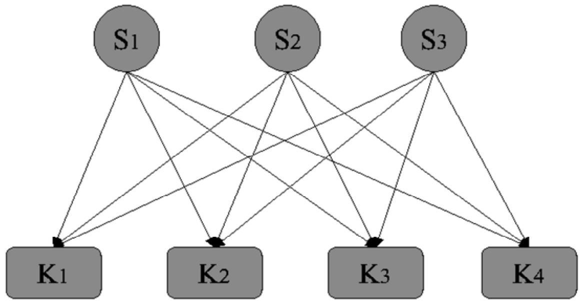

Figure 2. Abstract Overview of Grid System

The sources, upon knowing the amount of loads that they should give to each sink, will send their loads to all sinks simultaneously. Following Kim's model [2], we assume that the sinks will start computing the load fractions as they start receiving them. We also assume that the communication time delay is negligibly smaller than the computation time owing to high speed links so that no sinks starve for load.

We shall now introduce the definitions and notations that are used throughout this paper.

N (M) Total number of sources (sinks) in the system, with each source (sink) denoted by $S_i,\ i = 1, ...N$ ($K_j,\ j = 1, ...M$).

w_j Inverse of the computing speed of K_j.

L_i Load at S_i such that the total load in the system, $L =$ Sum (L_i , $\forall i = 1...N$).

$\alpha_{i,j}$ ($\hat{\alpha}_{i,j}$) Amount of load (estimated) that K_j shall request from S_i in an iteration.

α_j Fraction of load from $L^{(q)}$ that K_j should take in an iteration, α_j = Sum ($\alpha_{i,j}$, $\forall i = 1...N$).

T_{cp} Computing intensity constant.

$T^{(q)}$ Time taken to process the loads in the q-th iteration.

Y Fraction of the load L that should be taken into consideration in an iteration of installment, where $Y \leq 1$.

p Buffer estimator confidence factor.

$B_j^{(q)}$ ($\hat{B}_j^{(q)}$) Available (Estimated) buffer space in K_j in the q-th iteration.

P_{all} (P_{now}) Set of sinks (with buffer space available for processing in an iteration) in the system.

X_{now} Set of sources that are being processed in an iteration.

X_{new} Set of sources that arrive at the system when the system is idle or busy processing for some sources.

4. Dynamic Incremental Scheduling Strategy

We employ Kim's multi-port communication model [2] for load distribution and assume that K_1 generates the required schedule satisfying the resource constraints. Here, we also assume that the sinks will start computing the load fractions as they start receiving them and that the communication time delay is negligibly smaller than the computation time owing to high speed links so that no sinks starve for load. In the DLT literature [8], in order to derive an optimal solution it was mentioned that it is necessary and sufficient that all the sinks that participate in the computation must stop at the same time instant; otherwise, load could be redistributed to improve the processing time.

Using this optimality principle and assuming infinite buffer space at sink nodes, i.e., a sink can hold any amount of load from the sources, load fractions that a sink K_j shall receive from the source S_i is derived in [11] as

$$\alpha_{i,j} = \frac{1}{w_j(\sum_{x=1}^{M} \frac{1}{w_x})} L_i \qquad (1)$$

While deriving these load fractions, it is assumed that each sink requests a load fraction that is proportional to the size of the load at the source. Moreover, each sink requests the same load fraction (percentage of total load) from each source.

However, in real-life situations, each sink always has a limit to the amount of buffer space that can be used. Further, in a generic Grid environment, each node may be running multiple tasks such that it is required to share the available resources, hence there may be only a limited amount of buffer space that is allocated for processing particular loads at a given time. As a result, we are naturally confronted with the problem of scheduling divisible loads under buffer capacity constraints. The IBS algorithm proposed in [3] produces a minimum time solution given pre-specified buffer capacity constraints and it also exhibits finite convergence, but it does not consider scheduling under dynamic environments and buffer space variations at processing nodes. In this paper, we propose a Dynamic Incremental Scheduling Strategy (DISS) that takes into account the variations in buffer space availabilities with sink nodes and also propose an adaptive estimation scheme to schedule the processing loads in an incremental fashion.

In a real-life system, the total amount of loads to be processed may exceed the available buffer space at sinks and the loads may also arrive at arbitrary times to the system for processing. Thus, the number of loads to be processed may vary over time and also demand for processing may arise at

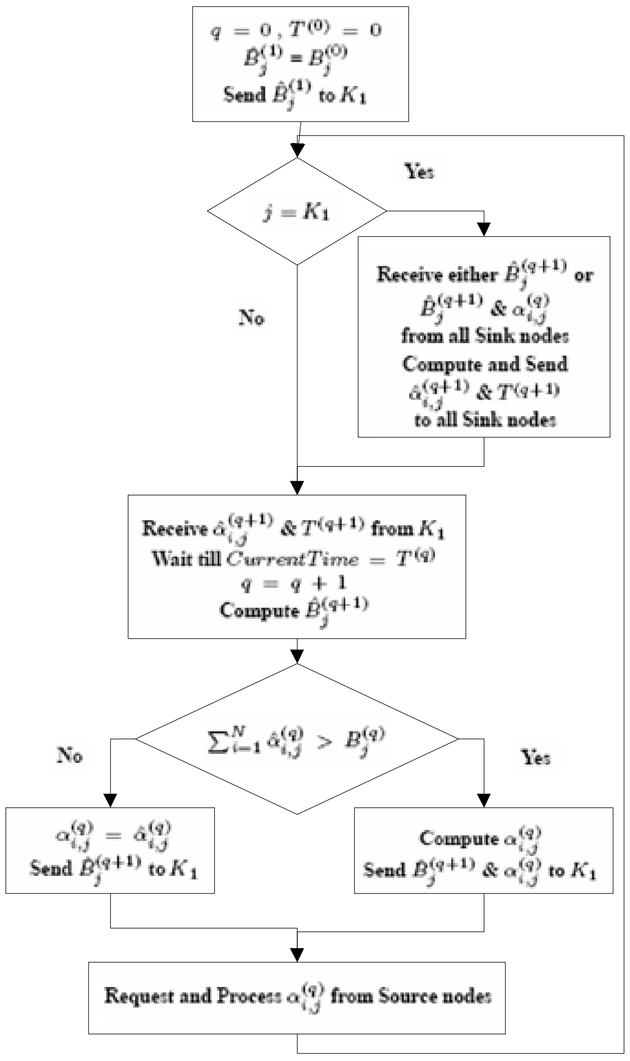
$q = 0$, $T^{(0)} = 0$
$\hat{B}_j^{(1)} = B_j^{(0)}$
Send $\hat{B}_j^{(1)}$ to K_1
$j = K_1$
Yes
No
Receive either $\hat{B}_j^{(q+1)}$ or $\hat{B}_j^{(q+1)}$ & $\alpha_{i,j}^{(q)}$ from all Sink nodes
Compute and Send $\hat{\alpha}_{i,j}^{(q+1)}$ & $T^{(q+1)}$ to all Sink nodes
Receive $\hat{\alpha}_{i,j}^{(q+1)}$ & $T^{(q+1)}$ from K_1
Wait till $CurrentTime = T^{(q)}$
$q = q + 1$
Compute $\hat{B}_j^{(q+1)}$
$\sum_{i=1}^{N} \hat{\alpha}_{i,j}^{(q)} > B_j^{(q)}$
No
Yes
$\alpha_{i,j}^{(q)} = \hat{\alpha}_{i,j}^{(q)}$
Send $\hat{B}_j^{(q+1)}$ to K_1
Compute $\alpha_{i,j}^{(q)}$
Send $\hat{B}_j^{(q+1)}$ & $\alpha_{i,j}^{(q)}$ to K_1
Request and Process $\alpha_{i,j}^{(q)}$ from Source nodes

Figure 3. Flowchart for a "Pull-based" DISS with Distributed Buffer Space Estimation

any time. It will be difficult to estimate a priori the maximum amount of load that may be in the system. This is especially true on Grids, as any node can attempt to inject a load when it has one to be processed. Under such conditions, a feasible scheduling may not exist unless the sink nodes allow their buffer space to be reclaimed after a given load is processed. This means that, after processing a given load, the sinks shall make their buffer space available for subsequent processing. Thus in order to handle the situation wherein sources demand processing at various time instants, dynamic scheduling strategies needs to be designed in such a way that sinks continue to render their available buffer space to the sources. The dynamic scheduling strate-

```
**Initial state:**
$I = \{1, 2, ...N\}$ , $J = \{1, 2, ...M\}$ , $q = 0$ , $T^{(0)} = 0$ , $p = 0.95$
$\hat{B}_j^{(1)} = B_j^{(0)}$ , $\alpha_{i,j}^{(0)} = 0$

**Step 1:** $K_1$ **computes** $\hat{\alpha}_{i,j}^{(q+1)}$ **&** $T^{(q+1)}$ **and communicates them to all Sink Nodes:**
$L_i = L_i -$ Sum $(\alpha_{i,j}^{(q)}, \forall j = 1...M)$ , $\forall S_i \in X_{now}$ , $i \in I$
If $(X_{new} \neq \emptyset)$ { $X_{now} = X_{now} \cup X_{new}$ , $X_{new} = \emptyset$ }
If $(L_i = 0)$ {$X_{now} = X_{now} - \{S_i\}$}, $\forall S_i \in X_{now}$ , $i \in I$
$P_{now} = P_{all}$
If$(\hat{B}_j^{(q+1)} = 0)$ $P_{now} = P_{now} - K_j$ , $\forall K_j$ , $j \in J$
$\alpha_j^{(q+1)} = \frac{1}{w_j * \text{Sum}\,(\frac{1}{w_x}, \forall x=1...M)}$ , $\forall K_j \in P_{now}$, $j \in J$
$L =$ Sum $(L_i, \forall i = 1...N)$ , $\forall S_i \in X_{now}$ , $i \in I$
$Y = \min\{\frac{\hat{B}_j^{(q+1)}}{\alpha_j^{(q+1)} * L}$ , $\forall K_j \in P_{now}\}$ , $j \in J$
If $(Y > 1)$ $\{Y = 1\}$
$\hat{\alpha}_{i,j}^{(q+1)} = Y * \alpha_j^{(q+1)} * L_i$ , $\forall S_i \in X_{now}$ , $i \in I$ , $\forall K_j$ , $j \in J$
$T^{(q+1)} =$ Sum $(\hat{\alpha}_{i,j}^{(q+1)}, \forall i = 1...N) * w_j * T_{cp}$ , $K_j \in P_{now}$
$K_1$ communicates $\hat{\alpha}_{i,j}^{(q+1)}$ & $T^{(q+1)}$ to all Sink Nodes

**Step 2: All Sink Nodes compute** $\hat{B}_j^{(q+1)}$ **&** $\alpha_{i,j}^{(q)}$ **and communicate them to** $K_1$**:**
All Sink Nodes wait till their $Current\ Time = T^{(q)}$
$q = q + 1$
$\hat{B}_j^{(q+1)} = \frac{\text{Sum}\,((k * B_j^{(k)}), \forall k=1...q)}{\text{Sum}\,(k, \forall k=1...q)} * p$
If ( Sum $(\hat{\alpha}_{i,j}^{(q)}, \forall i = 1...N) > B_j^{(q)}$) {
    $\alpha_{i,j}^{(q)} = \hat{\alpha}_{i,j}^{(q)} * \frac{B_j^{(q)}}{\text{Sum}\,(\hat{\alpha}_{i,j}^{(q)}, \forall i=1...N)}$
    Communicate $\hat{B}_j^{(q+1)}$ & $\alpha_{i,j}^{(q)}$ to $K_1$ }
else { $\alpha_{i,j}^{(q)} = \hat{\alpha}_{i,j}^{(q)}$
    Communicate $\hat{B}_j^{(q+1)}$ to $K_1$}

**Step 3: All Sink Nodes schedule the loads from Source Nodes:**
$B_j^{(q)} = B_j^{(q)} -$ Sum $(\alpha_{i,j}^{(q)}, \forall i = 1...N)$
Sink Nodes request and process load fractions $\alpha_{i,j}^{(q)}$ from Source Nodes
**Go to Step 1**
```

Figure 4. Pseudo code for a "Pull-based" DISS with Distributed Buffer Space Estimation

gies also need to take into account that the amount of buffer space available at sinks may vary over time and this variation may not be known a priori. Under such conditions, the sinks shall estimate the amount of buffer space that it could offer for scheduling in the next iteration and communicate it to the scheduler node. A buffer estimation strategy is described later in the Section 5. With this information, the scheduler node shall generate the required schedule satisfying the resource constraints. It may be noted that in order to estimate the load fractions $\hat{\alpha}_{i,j}$, the scheduler will use Equation 1.

As long as there is sufficient load in the system to completely consume the estimated amount of buffer space at one of the sink nodes, the load fractions $\alpha_{i,j}$ that a sink K_j may request from a source S_i has to be reduced by a factor Y, given by

$$Y = \min\left\{\frac{\hat{B}_j}{(\alpha_j L)}\right\} \tag{2}$$

This ensures that at each iteration all the sinks that participate in processing the loads complete processing at the same time instant, if the actual buffer space available at a sink node is equal to the estimated one at that node. The algorithm attempts to fill up one or more sinks' buffer space in every iteration. In any iteration, if the remaining load is not enough to completely consume the buffer space at any of the participating nodes, the suggested distribution by equation (1) will be used. When two sinks have identical buffer space, the one at the fastest sink will be fully utilized. As long as there is enough load to be processed in an iteration, the algorithm ensures that at least one sink's buffer is completely utilized. The processing time for the q-th iteration is given by,

$$T^{(q)} = \sum_{i=1}^{N} \hat{\alpha}_{i,j}^{(q)}\, w_j\, T_{cp} \tag{3}$$

In the above algorithm, the load fractions have been calculated based on the estimated buffer availabilities at the sinks. But, at the start of the next iteration, the actual buffer availabilities at the sinks may be different from the estimated values. As long as the load fractions assigned to each sink node by the node K_1 is less than or equal to the actual buffer availabilities at those sink nodes, the sink nodes can request for the load fractions assigned to them from the sources. But, if the buffer available at a sink is less than the load fraction assigned to it, then it could not process the excess load that has been assigned to it. Hence, those sinks shall recompute the load fraction as given by

$$\alpha_{i,j} = \hat{\alpha}_{i,j} * \frac{B_j}{\sum_{i=1}^{N} \hat{\alpha}_{i,j}} \tag{4}$$

and request these load fractions from the sources. In addition to requesting these load fractions from the sources, the sink node also has to communicate the modified load fractions that it has requested from the sources to the master sink node K_1. This is done for computing the exact amount of loads that remain at the sources for processing for the next iteration, by K_1. This information can be piggy backed along with the estimated buffer availability at the sink nodes that all sink nodes communicate to the master sink node K_1. Also, let us suppose that K_1 attempts to fill the entire buffer of K_2. Suppose if K_2 could not accommodate all the load assigned to it, then it modifies the value of $\hat{\alpha}_{i,j}$ assigned to it. Then K_2 together with other sink nodes that did not participate in processing in that iteration shall wait for all the other nodes to complete their processing (that is, for the time $T^{(q)}$) before requesting for loads from the sources again.

The optimal load fractions for the $(q+1)$-th iteration shall be estimated by the master sink node K_1 while processing the load for the (q)-th iteration, based on the total amount of load that remains to be processed. This process shall continue until all of the loads are processed. Note that in this case, the load requesting by the sinks and processing are dynamic in the sense that the IBS algorithm is invoked to estimate the load distribution depending on the number of sources and their respective load sizes. It may be noted that it is not necessary for a sink to render diminishing buffer space in every iteration to each source, since, the load to be processed from a source also diminishes. Further, it should be realized that the buffer space availability in sinks does not have an affinity towards any source. Thus, if no other sources demand processing, then the entire buffer is allocated to the demanding source. Figures 3 and 4 summarizes the above discussed policy.

The new set of loads and the unprocessed loads from the existing sources are considered together for scheduling in the next iteration. The following example clarifies the working principle of the above strategy. The parameters and data for this example are from real-life high energy nuclear physics experiments [12].

Example 1: Let us suppose that there are three sources with loads to be processed and there are four sinks that can process these loads. Let the speed parameter of sinks be $w_1 = 1.11 \times 10^{-9}$, $w_2 = 6.25 \times 10^{-10}$, $w_3 = 5.00 \times 10^{-10}$ and $w_4 = 3.57 \times 10^{-10}$, respectively. Let $T_{cp} = 6.52 \times 10^{12}$sec/load. Let the actual buffer capacities at sinks at the initial state (that is, iteration $q = 0$) and iteration $q = 1$ be $B_1^{(q)} = 6$, $B_2^{(q)} = 5$, $B_3^{(q)} = 0$, and $B_4^{(q)} = 2$; at iteration $q = 2$ be $B_1^{(2)} = 4$, $B_2^{(2)} = 3$, $B_3^{(2)} = 1$, and $B_4^{(2)} = 1$; at iteration $q = 3$ be $B_1^{(3)} = 2$, $B_2^{(3)} = 0$, $B_3^{(3)} = 2$, and $B_4^{(3)} = 1$; and at iteration $q = 4$ be $B_1^{(4)} = 1$, $B_2^{(4)} = 1$, $B_3^{(4)} = 3$, and $B_4^{(4)} = 1$ units respectively. These values are generated randomly using a uniform probability distribution in the range $[0, 7]$ in each iteration. We let the three sources to have loads $L_1 = 5$, $L_2 = 2$ and $L_3 = 3$ unit loads, respectively. Let loads L_1 and L_2 arrive at $t = 0$ seconds, and load L_3 arrive at $t = 5 \times 10^3$ seconds. Note that the computationally intensive nature of the problem is reflected by the parameter T_{cp}. Using the above algorithm, we have the values for $\alpha_{i,j}^{(q)}$ as shown in Tables 1 and 2. The unutilized buffer space in all the iterations is given in the last column of Table 1. From, these results, we observe that buffer of K_4 is fully utilized in iterations 1 and 2, whereas buffer of K_3 is not at all utilized in iteration 2 (because estimated buffer size is 0 for that iteration). For iteration 3, buffer of K_3 is estimated to be less than the actual value and hence buffers of all the available sinks are under utilized in that iteration. At the final iteration, the remaining load is insufficient to completely fill up the buffer at any of the sinks. The distribution suggested by the values $\alpha_{i,j}$ in the Table 2 are used by the sinks. Iteration 1 to 4 are scheduled at time $t = 0,\ 4.655 \times 10^3,\ 8.607 \times 10^3,$ and 1.067×10^4 seconds, respectively. The total processing time for processing all the three loads is $t = 1.236 \times 10^4$ seconds. From this example, it is seen that, because of the new source S_3 and the buffer space variations at the sinks, the processing time for the other sources in the system is stretched to $t = 1.236 \times 10^4$ seconds. Below we describe the buffer estimation strategy and its impact on the performance with respect to this example is discussed in Section 6.

$q=1$	$\sum \hat{\alpha}_{i,j}^{(1)}$	$\sum \alpha_{i,j}^{(1)}$	$\hat{B}_j^{(1)}$	$B_j^{(1)}$
K_1	0.643	0.643	6.000	5.357
K_2	1.143	1.143	5.000	3.857
K_3	0.000	0.000	0.000	0.000
K_4	2.000	2.000	2.000	0.000
$q=2$	$\sum \hat{\alpha}_{i,j}^{(2)}$	$\sum \alpha_{i,j}^{(2)}$	$\hat{B}_j^{(2)}$	$B_j^{(2)}$
K_1	0.546	0.546	5.700	3.454
K_2	0.970	0.970	4.750	2.030
K_3	0.000	0.000	0.000	1.000
K_4	1.698	1.000	1.900	0.000
$q=3$	$\sum \hat{\alpha}_{i,j}^{(3)}$	$\sum \alpha_{i,j}^{(3)}$	$\hat{B}_j^{(3)}$	$B_j^{(3)}$
K_1	0.284	0.284	4.433	1.716
K_2	0.506	0.000	3.483	0.000
K_3	0.633	0.633	0.633	1.367
K_4	0.886	0.886	1.267	0.114
$q=4$	$\sum \hat{\alpha}_{i,j}^{(4)}$	$\sum \alpha_{i,j}^{(4)}$	$\hat{B}_j^{(4)}$	$B_j^{(4)}$
K_1	0.235	0.235	3.167	0.765
K_2	0.415	0.415	1.742	0.585
K_3	0.518	0.518	1.267	2.482
K_4	0.727	0.727	1.108	0.273

Table 1. Buffer utilization values

$q = 1$	S_1	S_2		$\sum \alpha_{i,j}^{(1)}$
K_1	0.459	0.184		0.643
K_2	0.817	0.326		1.143
K_3	0.000	0.000		0.000
K_4	1.429	0.571		2.000

$q = 2$	S_1	S_2		$\sum \alpha_{i,j}^{(2)}$
K_1	0.390	0.156		0.546
K_2	0.693	0.277		0.970
K_3	0.000	0.000		0.000
K_4	0.714	0.286		1.000

$q = 3$	S_1	S_2	S_3	$\sum \alpha_{i,j}^{(3)}$
K_1	0.038	0.015	0.231	0.284
K_2	0.000	0.000	0.000	0.000
K_3	0.085	0.034	0.514	0.633
K_4	0.119	0.048	0.719	0.886

$q = 4$	S_1	S_2	S_3	$\sum \alpha_{i,j}^{(4)}$
K_1	0.032	0.013	0.190	0.235
K_2	0.056	0.022	0.337	0.415
K_3	0.070	0.028	0.420	0.518
K_4	0.098	0.040	0.589	0.727

Table 2. Values for load fractions

5. Buffer estimation Strategy

We propose a distributed buffer estimation strategy based on weighted average calculations. The weights for computing the estimates are based on the iteration indices until the current iteration. We refer to this estimator as *Iteration Index based Buffer estimator* (IIB). Our IIB 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 master sink node K_1. Then, K_1 shall execute the dynamic scheduling algorithm described in Figure 4 to determine the $\hat{\alpha}_{i,j}$ that the participating sink nodes shall request from the sources.

For estimating the buffer availability at a sink, each sink K_j needs to keep track of the actual buffer sizes B_j from its previous iterations. Note that for implementation purposes it is sufficient to keep a cumulative value for the weighted buffer space. In any iteration q, each sink node shall estimate the buffer size that will be available for the next iteration $(q+1)$ as

$$\begin{aligned} \hat{B}_j^{(q+1)} &= \left\{ \frac{\sum_{k=1}^{q}((k/q) * B_j^{(k)})}{\sum_{k=1}^{q}(k/q)} \right\} * p \\ \Rightarrow \\ &= \left\{ \frac{\sum_{k=1}^{q}(k * B_j^{(k)})}{\sum_{k=1}^{q} k} \right\} * p \end{aligned} \tag{5}$$

and declare it to the master sink node. In Equation 5, 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%, for the next iteration. This may be observed in our example as discussed in Section 6.

6. Discussions and Conclusions

The contributions in this paper are geared towards designing and analyzing a dynamic scheduling strategy for handling large volume loads that arrive to a grid system for processing. The strategy that we proposed in this paper is suitable for handling large scale data generated in physics experiments (as discussed in Section 1). Since a Grid infrastructure is always viewed as a repository of resources that can be availed by careful scheduling, implicit to this problem are some real-life constraints such as availability of the nodes for processing, the amount of resources they can render, speeds with which the nodes and links can respond etc. Also, as in the case of any networked system, here too, we can follow a “pull-based” or “push-based” approaches. In this paper, we considered a pull-based strategy. Further, we considered a real-life situation where in the sinks have finite sized buffer and hence the available buffer space have to be shared in an optimal manner among the competing sources. Also, we assumed that every sink attempts to request loads from all participating sources for processing.

We tuned the IBS algorithm proposed in the literature to tackle the posed problem. In addition, since the availability of buffer spaces is dynamic, we proposed an estimation strategy IIB which works on weighted average values as explained. The impact of IIB with respect to Example 1 is as follows. In Table 1 we show the estimated as well as the actual loads requested by the sinks. Further we also project the estimated buffer values. Following are important points to observe. In iteration 1, the estimated and the actual loads being same, the buffer rendered is adequate to handle the estimated load. However, this does not carry far in the second iteration. In iteration 2, we observe that at K_4, the estimated load being more than the actual buffer rendered, the actual load that is to be requested is tailored to adapt to the available space. It may also be observed that in iteration 2, the estimated buffers take into account the actual buffers rendered in the past iteration. This will be cumulatively done in each iteration, which is indeed the essence of our design. Further, in iteration 2, the actual buffer available is unutilized, as the estimated value is 0. This prevents K_3 to request any load from the source at this iteration. This is somewhat natural to expect which is captured in our design. Another important observation comes from the fact that in iteration 2, if the estimated load sizes have been requested by all the sinks then the sources S_1 and S_2 could have been completely processed in this iteration itself. However since K_4 could not accommodate the estimate load, S_1 and S_2 are forced to be considered for scheduling in the future iterations as well.

Also, note that although S_3 becomes available for processing after iteration 2 starts, it is considered for processing in iteration 3 onwards. Note that in iteration 3, the estimated buffer at K_3 is observed to be less than the actual buffer available. Thus, the scheduler considers a load based on a minimum of the actual or estimated buffer space. In our case, this turns out to be the estimated buffer value. Now, when the estimated total load to be processed is less than or equal to the available buffer spaces, then all the loads could be scheduled and processed at this iteration itself. This happens at iteration 4 in our example.

The proposed IIB strategy works as long as the buffer variations are not drastic. Further, if new loads arrive to the system before the loads being processed are completed, then the processing of existing loads will be stretched. Thus, when loads to be processed are not time-critical the strategy is highly recommended. This aspect indeed triggers an open issue for refining this approach to accommodate an admission control mechanism that could adapt to random arrivals of the loads. This is one of our future extensions.

Acknowledgments

Thomas Robertazzi's research is funded by NSF grant CCR-99-12331. Dantong Yu's research is supported by DOE PPDG/ATLAS/RHIC grants. Sivakumar Viswanathan's research is supported by Institute for Infocomm Research, Singapore.

References

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

[2] H.-J. Kim. A Novel Optimal Load Distribution Algorithm for Divisible Loads. Kluwer Academic Publishers, January 2003.

[3] X. Li, B. Veeravalli, and C. Ko. Divisible Load Scheduling on Single Level Tree Networks with Buffer Constraints. *IEEE Transactions on Aerospace and Electronic Systems*, 36(4):1298–1308, Oct. 2000.

[4] T. Robertazzi. Ten Reasons to Use Divisible Load Theory. *Computer*, 2003.

[5] A. Rosenberg. Sharing Partitionable Workload in Heterogeneous NOWs: Greedier is Not Better. In *Proc. of IEEE International Conference on Cluster Computing*, pages 124–131, Newport Beach CA, USA, 2001.

[6] J. Sohn and T. Robertazzi. An Optimal Load Sharing Strategy for Divisible Jobs with Time-Varying Processor Speeds. *IEEE Transactions on Aerospace and Electronic Systems*, 34(3):907–923, July 1998.

[7] B. Veeravalli and G. Barlas. Scheduling Divisible Loads with Processor Release Times and Finite Size Buffer Capacity Constraints. In T. G. Robertazzi and D. Ghose, editors, *special issue of Cluster Computing on Divisible Load Scheduling*, volume 6 of *1*, pages 63–74. Kluwer Academic Publishers, Jan. 2003.

[8] B. Veeravalli, D. Ghose, V. Mani, and T. Robertazzi. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press, Los Alamitos, CA, Sept. 1996.

[9] B. Veeravalli, D. Ghose, and T. G. Robertazzi. Divisible Load Theory: A New Paradigm for Load Scheduling in Distributed Systems. In T. G. Robertazzi and D. Ghose, editors, *special issue of Cluster Computing on Divisible Load Scheduling*, volume 6 of *1*, pages 7–18. Kluwer Academic Publishers, Jan. 2003.

[10] S. Viswanathan, B. Veeravalli, D. Yu, and T. G. Robertazzi. Pull-based resource aware scheduling on large-scale computational grid systems. Technical Report TR/OSSL/VB/GC-01-2004.

[11] H. Wong, D. Yu, B. Veeravalli, and T. Robertazzi. Data Intensive Grid Scheduling: Multiple Sources with Capacity Constraints. In *IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2003)*, Marina del Rey, CA, Nov. 2003.

[12] D. Yu and T. Robertazzi. Divisible Load Scheduling for Grid Computing. In *IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2003)*, Marina del Rey, CA, Nov. 2003.