The Journal of Supercomputing, 34, 135–163, 2005

Characterization of Bandwidth-Aware Meta-Schedulers for Co-Allocating Jobs Across Multiple Clusters

WILLIAM M. JONES wjones@parl.clemson.edu
WALTER B. LIGON III walt@parl.clemson.edu
LOUIS W. PANG plouis@parl.clemson.edu
Parallel Architecture Research Lab, Department of Electrical and Computer Engineering, Clemson University, 105 Riggs Hall, Clemson, SC 29634-0915

DAN STANZIONE dstanzi@asu.edu
High Performance Computing Center, Fulton School of Engineering, Arizona State University, P.O. Box 875206, Tempe, AZ 85287-5206

Abstract. In this paper, we present a bandwidth-centric job communication model that captures the interaction and impact of simultaneously co-allocating jobs across multiple clusters. We compare our dynamic model with previous research that utilizes a fixed execution time penalty for co-allocated jobs. We explore the interaction of simultaneously co-allocated jobs and the contention they often create in the network infrastructure of a dedicated computational multi-cluster.

We also present several bandwidth-aware co-allocating meta-schedulers. These schedulers take inter-cluster network utilization into account as a means by which to mitigate degraded job run-time performance. We make use of a bandwidth-centric parallel job communication model that captures the time-varying utilization of shared inter-cluster network resources. By doing so, we are able to evaluate the performance of multi-cluster scheduling algorithms that focus not only on node resource allocation, but also on shared inter-cluster network bandwidth.

Keywords: parallel job scheduling, multiple clusters, bandwidth-aware, network contention, job co-allocation, multi-site scheduling, simulation

1. Introduction

Clusters of commodity processors have become fixtures in research laboratories around the world. Collections of several co-located clusters exist in many larger laboratories, universities, and research parks. This co-location of several resource collections naturally lends itself to the formation of a multi-cluster (Figure 1).

A multi-cluster is distinguished from a traditional computational grid in that the multi-cluster utilizes a dedicated interconnection network among cluster resources with a known topology and predictable performance characteristics. This type of networking infrastructure allows for the possibility of mapping jobs across cluster boundaries in a process known as *co-allocation* or *multi-site scheduling* (Figure 2).

In this paper, we develop a parallel job model that takes both computation and communication into account as a means by which to explore co-allocating multi-cluster schedulers

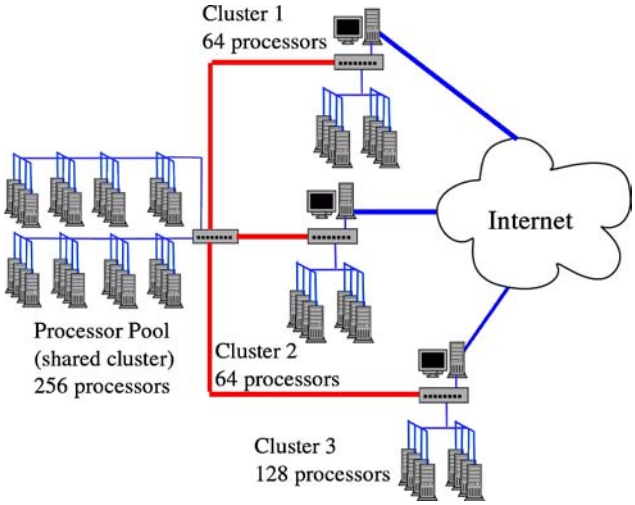

Figure 1. A multi-cluster.

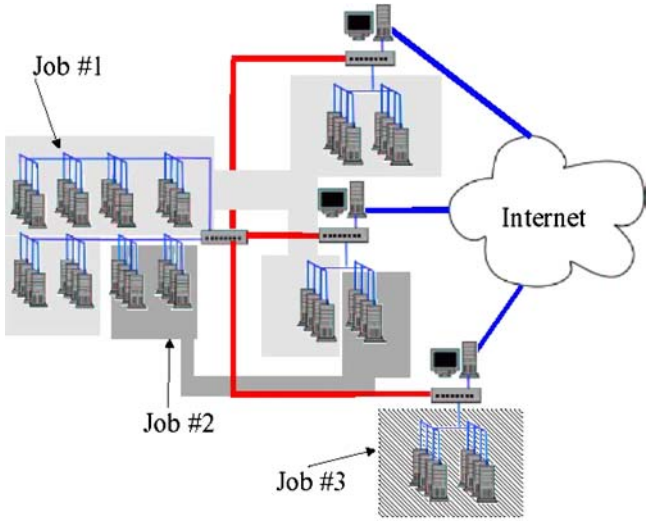

Figure 2. Job co-allocation.

that exploit these unique architectural features [8]. We present an in-depth explanation of our communication model and its associated algorithms as well as a study of the impact of co-allocation in a multi-cluster as a function of job communication characteristics and scheduling routines. We compare our dynamic model with previous research that utilizes a fixed execution time penalty for co-allocated jobs.

We also develop several bandwidth-aware co-allocating meta-schedulers that take inter-cluster network utilization into account as a means by which to mitigate the slowdown

associated with the interaction of simultaneously co-allocating jobs across multiple clusters [7]. By making use of a bandwidth-centric parallel job communication model that captures the time-varying utilization of shared inter-cluster network resources, we are able to evaluate the performance of multi-cluster scheduling algorithms that focus not only on computational resource allocation, but also on shared inter-cluster network bandwidth. Lastly, we provide an in-depth explanation of our bandwidth-aware co-allocation algorithms and compare and contrast their performance characteristics.

Previous work in the area of job co-allocation tends to characterize jobs by either specifying that all communications require a fixed amount of time to travel between clusters [1, 2] or by assigning co-allocated jobs a fixed execution-time penalty [3, 4]. This type of characterization is not sensitive to the time-varying contention for bandwidth in the inter-cluster communication links and the impact it has on the execution time of co-allocated jobs that share network resources. We take a different approach by considering that as jobs become co-allocated or co-allocated jobs terminate, there is a continual change in the available inter-cluster bandwidth. Therefore, in our work, the duration of wide area communication is a function of the time-varying network bandwidth utilization among clusters participating in the multi-cluster, which in turn affects the execution time of co-allocated jobs. This research aims to extend the work presented in [1] and [2] by replacing the static communication model with a more dynamic view of job communication that is *bandwidth-centric*.

We find that schedulers designed to allocate node resources across cluster boundaries can result in rather poor overall performance over a wide range of workload characterizations and multi-cluster configurations due to the interaction simultaneously co-allocated jobs experience as they contend for inter-cluster network bandwidth. Our research therefore focuses on a range of algorithms with varying levels of complexity that attempt to mitigate this impact.

2. Computational multi-clusters and meta-scheduling

At first glance, multi-clusters may appear to be distinguished from conventional computational grids only in scope. A multi-cluster is limited to a campus-wide setting, for example, while a traditional grid is national or even global in scope. However, upon further inspection, a multi-cluster has a distinctive architectural feature; the internal networks of the clusters are bridged together through dedicated links. This has several important implications. First, there exists predictable, reliable bandwidth between cluster resources, as contrasted with Internet connected grids. This should allow a scheduler to make better decisions in co-allocating jobs across these resources.

Additionally, finer grain control of resources is more practical within the multi-cluster framework. When fast, low latency links are available, reallocating sets of nodes from one cluster to another is a low overhead operation.

In order to efficiently leverage the collective computational power of a multi-cluster, special scheduling agents are required to select and maps jobs to available resources. We refer to these schedulers as *meta-schedulers* (Figure 3). In general, we consider a meta-scheduler to be the software, or collection of software, that decides where, when, and how to schedule jobs in a multi-cluster. A meta-scheduler is expected to work in conjunction

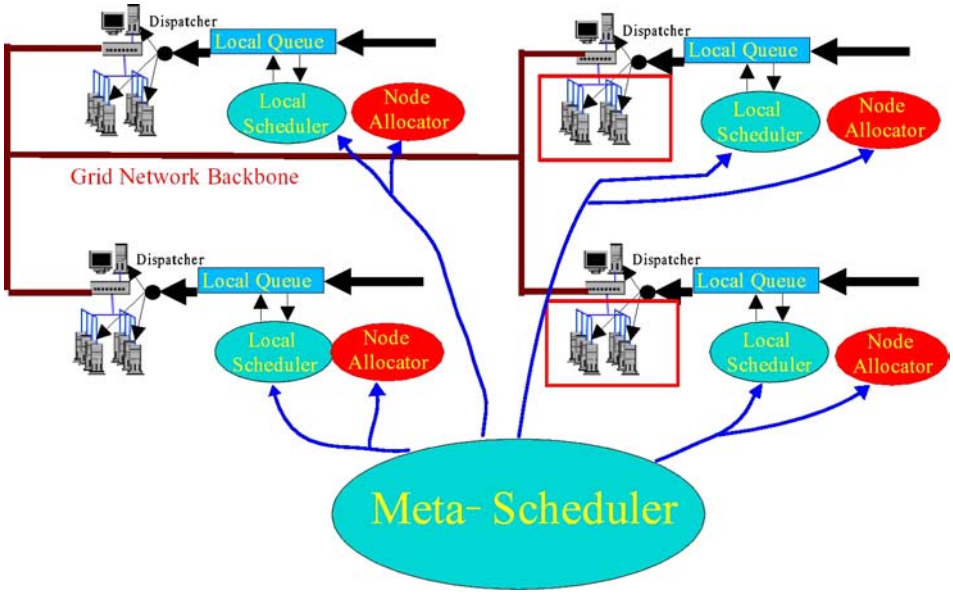

Figure 3. Meta-scheduler.

with the local schedulers working on each individual cluster. In this paper, we assume that the meta-scheduler is globally aware of the state of the multi-cluster.

The initial motivation for this research was an interest in developing new scheduling and resource management software for our own computational multi-cluster (Figure 1). A discrete event-driven simulator, known as *Beosim* (Section 3.8), was developed in order to study the effects of various scheduling routines and explore the behavior of a multi-cluster under a variety of workload characterizations.

3. The model

In this section we characterize the parallel job model as well as the multi-cluster architecture. We provide a detailed explanation of the communication model used, as well as a strategy to account for the time-varying inter-cluster network utilization. Additionally, we provide a brief description of our custom multi-cluster simulator.

3.1. Multi-cluster model

In the research presented in this paper, we consider a multi-cluster to be a collection of arbitrary sized clusters with globally homogeneous nodes. Each cluster has its own internal ideal switch. Additionally, the clusters are connected to one another through a single dedicated link to a central ideal switch (Figure 4). Each node in the multi-cluster has a single

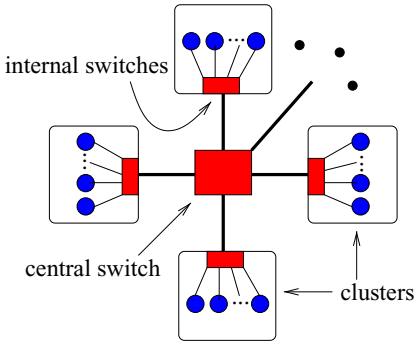

Figure 4. Multi-cluster topology.

processor and a single network interface card. Jobs can be co-allocated in a multi-cluster by allocating nodes from different clusters to the same job in order to better meet collective needs across the multi-cluster.

3.2. *Parallel job model*

The model used assumes that jobs are non-malleable. In other words, each job requires a fixed number of processors for the life of the job, and the scheduler may not adjust this number. Additionally, neither execution-time migration nor gang-scheduling [12] is employed in mapping the job onto the multi-cluster, i.e. once the job is mapped to a particular set of nodes, the job remains on these nodes for the lifetime of its execution.

A job's execution time, T_E, is a function of two components, the computation time, T_P, and the communication time, T_C. The initial value of these two quantities is considered to represent the total execution time that the job would experience on a *single dedicated cluster* with an ideal switch. They therefore form a basis for the best-case execution time of a given job when it is co-allocated in the multi-cluster. In particular, $T_E = T_P + T_C$. The computation portion of the execution time does not vary, however the communication time is considered dynamic, since the communication time of simultaneously co-allocated jobs may be lengthened due to the utilization of any shared inter-cluster network links.

3.3. *Communication characterization*

Each job modeled in this study performs all-to-all global communication patterns periodically throughout its execution. Each node in a given job j, is characterized by an average per-processor bandwidth requirement, $PPBW_j$, that consists of the bandwidth needed to both send and receive all messages associated with a node. During co-allocation, nodes must communicate across cluster boundaries. This communication will require a certain about of bandwidth in the inter-cluster network links. A job's performance will deteriorate if it does not receive the amount of bandwidth it requires to run at full speed. In order to

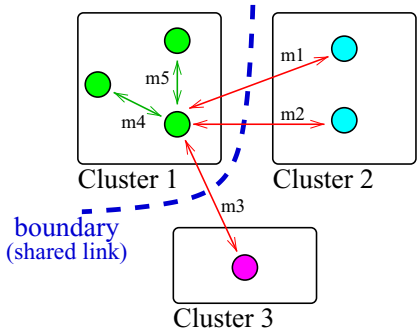

Figure 5. Bandwidth calculation example.

determine when the inter-cluster links become saturated, we must first identify how much bandwidth a job will require in order to run at full speed. The amount of bandwidth, BW_i^j, required by job j on inter-cluster link i is given by Equation (1), where n_T^j is the total number of nodes required by job j and n_i^j is the number of nodes allocated to job j on cluster C_i. This equation is based on all-to-all communication, which is assumed to dominate the communication time of the program.

$$BW_i^j = \left(n_i^j * PPBW_j\right)\left(\frac{n_T^j - n_i^j}{n_T^j - 1}\right) \tag{1}$$

The first factor is total bandwidth required by all the nodes associated with job j on cluster C_i. The second factor in this equation represents the fraction of the messages generated by each node on cluster, C_i, that are destined for non-local nodes. For example, suppose that a job consists of six total nodes and has been mapped onto a multi-cluster consisting of three clusters, as shown in Figure 5. The total bandwidth required by all nodes local to cluster 1 would be $(3 * PPBW)$, since there are three nodes local to cluster 1. For each all-to-all communication, each of the three nodes on cluster 1 will generate five messages, i.e. $(6-1)*3 = 15$ total messages. Of these 15 messages, only $(6-3)*3 = 9$ will traverse cluster 1's inter-cluster network link. The ratio of the number of messages traversing cluster 1's network link to the total number of messages represents the percentage of the total bandwidth that is required by this job on cluster 1's inter-cluster network link, e.g.

$$BW_1^j = (3 * PPBW_j)\left(\frac{(6-3)*3}{(6-1)*3}\right). \tag{2}$$

Figure 5 depicts the messages sent and received by a single node on cluster 1; however in practice, all nodes send and receive messages.

Once a job has been mapped to the multi-cluster, the required bandwidth, BW_i^j, is calculated for each link, i. This amount is then aggregated with the bandwidth required by every other job that shares this link. Using this quantity, the co-allocated jobs' residual times

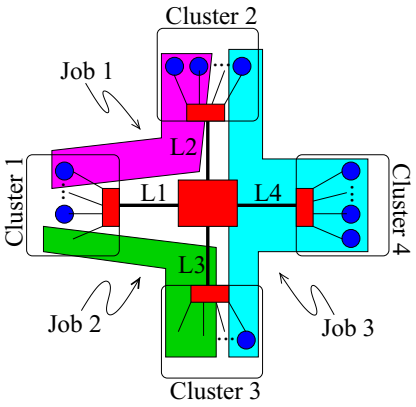

Figure 6. Co-allocation example.

to completion are recalculated for each event that causes a state change in any inter-cluster links. The details for these recalculations are provided in the subsections that follow.

3.4. Job co-allocation

We consider co-allocation (Figure 6) to be the mapping of a job across cluster boundaries. One possible reason that a job might be co-allocated is due to the natural fragmentation that occurs within each cluster in the node dimension. Suppose for example that a job is waiting in a cluster's ready queue. This job may require more nodes than are presently available on its particular cluster, but collectively there may be enough available nodes elsewhere in the multi-cluster to accommodate the job. The job would be considered co-allocated if it were mapped onto nodes that were "borrowed" from other clusters.

3.5. Intra-cluster bandwidth saturation

When jobs are co-allocated, their effective execution times may be altered due to the inherent bottleneck that is created in the multi-cluster's interconnection network. The degree to which the job's runtime is affected depends on several factors. Clearly the time-varying utilization of the dedicated links connecting the clusters plays an important role. Additionally, the amount of communication that each co-allocated job produces can considerably affect not only its own execution time, but also that of every other co-allocated job that shares any network resources with it.

The first step in determining the impact of co-allocation is to identify the presence and location of communication bottlenecks in the inter-cluster links. The residual time to completion for a particular job can change in response to two events, in particular, when a new job is co-allocated in the multi-cluster, or when a co-allocated job terminates and thus frees network resources.

Each inter-cluster link, i, is characterized by a maximum bandwidth rating, $BW_i^{\max}$. An initial measure of the saturation of each link is calculated by taking the ratio of the maximum available bandwidth to the total bandwidth required for every job that spans that link. The saturation ratio is given by Equation (3)

$$BW_i^{\text{sat}} = \frac{BW_i^{\max}}{\sum_{\forall j \in J_i} BW_i^j} \tag{3}$$

where set J_i is the set of all jobs that span link i. If $BW_i^{\text{sat}} \geq 1.0$ then link i is *not* saturated, otherwise if $(0.0 \leq BW_i^{\text{sat}} < 1.0)$, then link i is saturated. If a given link i is saturated, then each job in J_i will not be able to receive the amount of bandwidth it requires to run at full speed. In order to calculate the impact on each job due to co-allocation, the fraction of bandwidth each job receives compared to the amount it requires must be determined.

Each time a new job is co-allocated or when a co-allocated job terminates, the algorithm below is applied in order to determine the amount of bandwidth ultimately alloted to each job on each link. In the following algorithm, Equation (4) is used to account for the residual saturation level of the inter-cluster links due only to the jobs that have not yet been constrained.

$$BW_i^{\text{uc\_sat}} = \frac{BW_i^{\text{avail}}}{\sum_{\forall j \in J_{\text{uc}}} BW_{(i,j)}^{\text{alloted}}} \tag{4}$$

Step 1: *Initialization*—For every job j, let $BW_{(i,j)}^{\text{alloted}} = BW_i^j$. For every link i, let $BW_i^{\text{avail}} = BW_i^{\max}$. Let the unconstrained set of nodes, $J_{uc} = J$ (set of all jobs). Let the set, J_i be the set of all jobs that span link i.

Step 2: *Saturation detection*—For every link, calculate $BW_i^{\text{uc\_sat}}$. While there exists at least one $BW_i^{\text{uc\_sat}} < 1.0$, continue, else goto Step 5.

Step 3: *Saturation correction*—Identify the link with the smallest $BW_i^{\text{uc\_sat}}$ (most saturated link) from Step 2, and globally reduce the alloted bandwidth of every job in $J_i \cap J_{uc}$ by a factor of BW_i^{sat}.

Step 4: *Update state*—Remove each of the modified jobs from the set J_{uc}. For each of the modified jobs, remove their alloted bandwidth from the available bandwidth, BW_i^{avail} on each link over which they span. Goto Step 2.

Step 5: *Termination*—DONE.

After this algorithm is applied, the alloted bandwidth, $BW_{(i,j)}^{\text{alloted}}$, for each job will either be its initially requested bandwidth, BW_i^j for full speed execution, or it will be some fraction of its required bandwidth (Figures 6 and 7). If the job is alloted its required bandwidth, it will not experience any slowdown associated with communication for the duration of time between the current event and the next inter-cluster state changing event. However, if the job does not receive its required bandwidth, it will experience a slowdown in its residual communication time that is proportional to the disparity between its required and alloted inter-cluster bandwidths.

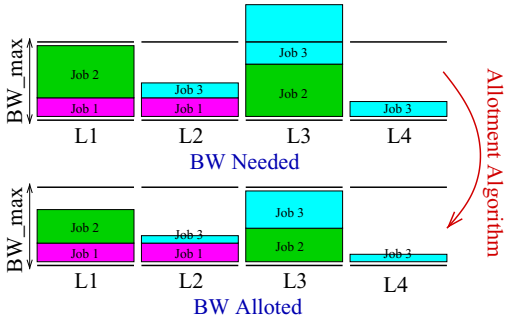

Figure 7. Bandwidth allotment.

3.6. *Co-allocated job communication slowdown*

Each affected job's bandwidth allotment on each link over which it spans is reduced in order to accommodate it's most saturated link. This bottleneck uniquely determines the disparity between the job's required and alloted bandwidths on each link. This implies that a job's ratio of the alloted to required bandwidth is the same for each link over which the job spans. Equation (5) formalizes the bandwidth slowdown associated with job j, where link k may be any link over which the job spans.

$$BW^{sd}_{(k,j)} = \frac{BW^{\text{alloted}}_{(k,j)}}{BW^{j}_{k}} \tag{5}$$

3.7. *Residual execution times*

Now that the communication slowdown factor is known, the residual execution time, T^R_E, of a job can be calculated as a function of both the residual communication and computation times (T^R_C and T^R_P respectively). Its associated end-event can then be rescheduled in the simulator to account for the state change in the inter-cluster network. In particular, Equations (6) and (7) illustrate the calculation required to determine the residual execution time of job j, where the primed terms represent quantities from the previous inter-cluster state changing event, while the non-primed values represent quantities for the current state change event.

$$T^R_E = \overbrace{\left(T^{R'}_C - T^{\Delta}_C\right)\left(BW^{sd'}_{(k,j)}\right)\left(BW^{sd}_{(k,j)}\right)^{-1}}^{T^R_C} + \overbrace{\left(T^{R'}_P - T^{\Delta}_P\right)}^{T^R_P} \tag{6}$$

where

$$T^{\Delta}_P = \frac{\Delta T}{T^{R'}_E} T^{R'}_P, \quad T^{\Delta}_C = \frac{\Delta T}{T^{R'}_E} T^{R'}_C \tag{7}$$

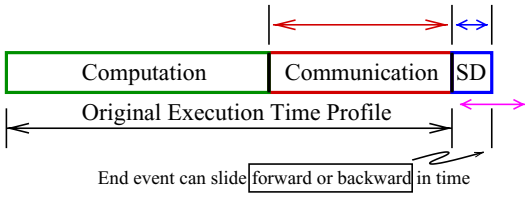

Figure 8. Bandwidth slowdown effect.

The T_P^{Δ} and T_C^{Δ} terms represent the times spent doing computation and communication respectively during the interval since the last state change event. These quantities can then be subtracted from the previous residual computation and communication times. The communication term is then scaled to take into account the slowdown due to its most saturated inter-cluster network link, as seen in Equation (6).

When a job is initially co-allocated, its residual computation (T_P^R), communication (T_C^R), and execution (T_E^R) times are initialized to T_P, T_C, and T_E respectively from its original profile. As inter-cluster state changing events occur, the residual times are recalculated based on Equations (6) and (7). Due to these recalculations, the job's end-event can slide forward (later) or backward (earlier) in time (Figure 8), reflecting either a degradation or improvement in saturation levels of the inter-cluster links over which it spans.

This procedure provides a dynamic view of job communication by accounting for the slowdown a job experiences due to the time-varying utilization of the inter-cluster network links.

3.8. Simulator

Beosim [8] is a discrete event driven simulator designed to model a multi-cluster as a collection of (possibly heterogeneous) computational clusters connected via a dedicated interconnection network. Beosim can be driven via synthetic workload distributions that are characterized through the use of randomly generated arrival and service processes.

Although Beosim has the capability of ingesting actual workload trace-files, in this paper we make use of synthetically generated workloads. In particular, we assume that the the arrival process of jobs to each cluster, C_i, has a Poisson distribution with rate λ_i. Additionally, we assume that a job's initial service time, T_E is exponential with parameter $(\mu_i)^{-1}$. The number of nodes that a job requires is given by a uniform distribution $D_i^{\text{nodes}} \sim UNIF[n_1^i, n_2^i]$. The fraction of *the total execution time* that initially represents computation is set to a constant, K_i, for all jobs.

4. Dynamic model comparison

In order to compare the results obtained from our dynamic communication model with that of previous research that utilizes a fixed penalty for co-allocated jobs [3, 4], we have

implemented several scheduling algorithms that can be used to evaluate our communication model and the impact of co-allocating jobs across multiple clusters.

4.1. Job selection policy

Each of the strategies described in this paper uses the classic **F**irst-**C**ome-**F**irst-**S**erved (FCFS) job selection policy with a slight modification. Specifically, the queue is traversed from head to tail looking for the first job that will fit into the available node sets. This policy is known as **F**it-**P**rocessors-**F**irst-**S**erved (FPFS) [10]. Traditionally a policy commonly known as *EASY backfilling* [9, 11] is used in many production grid schedulers, such as Maui [6]. This method will attempt to run jobs in FCFS order, but in the event that the job at the head of the queue *cannot* run due to insufficient resources, it will traverse the queue from head to tail searching for the first job that *can* run given the currently available free resources, provided that by doing so, the start time of the job at the head of the queue is not delayed. This technique is typically used as a means by which to provide a degree of flexibility in backfilling node-time holes in the schedule, while guaranteeing that no starvation takes place.

By making use of the bandwidth-centric communication model, only an estimate of a job's end event is known at any instant in time, since a job's end event can slide forward and backward in time depending on the communication contention in the inter-cluster network links. This makes it difficult to guarantee that the highest priority job's reservation in EASY backfilling will be meet, since we do not terminate jobs; therefore, we do not employ EASY backfilling.

4.2. Co-allocation strategies

The *First-Fit Strategy* performs job co-allocation by assigning node resources starting with the cluster with the largest number of free nodes. It then spans as many clusters as necessary to satisfy the job's node requirement. By employing this technique, the number of inter-cluster links over which a given job will span is minimized.

In order to establish an "reasonable" upper and lower bound for the job turnaround time metric, three baseline simulations were conducted to identify these levels. The first is run under the assumption that the inter-cluster network links have unlimited bandwidth capacities. This configuration, *Ideal*, represents a "best-case" that can be seen as a lower bound for average job turnaround time, since there is no slowdown associated with job co-allocation. The second strategy is referred to as *Migration Only*. This strategy only performs job migration, i.e. no job co-allocation. Jobs that are migrated do not contend for inter-cluster network resources. Therefore their ultimate execution times are also unaffected by their bisection bandwidth. Additionally, a third bound, *No Share*, is included that represents the the performance of the multi-cluster when all jobs that arrive to a given cluster *must* run must run locally. In this configuration, no resource sharing can take place.

The *Ideal*, *Migration Only*, and *No Share* strategies therefore appear as horizontal "limits" in the included figures, since they are unaffected by a job's bisection bandwidth.

4.3. Experimental setup

The first set of experiments were conducted using the following parameters. Each cluster in the multi-cluster consists of 100 homogeneous computational nodes and has a 1000 Mbps inter-cluster network link to the central switch. The workload presented to each cluster consists of 4,000,000 jobs. Such a large number of jobs were required in order to achieve convergence in the job turnaround time performance metric [5]. The number of nodes each job requires is taken from a uniform distribution *UNIF* [10, 90] (nodes). The job arrival process is Poisson with the inter-arrival time taken from an exponential distribution with parameter 150 (sec). The base execution time of each job is taken from an exponential distribution with parameter 225 (sec). In order to restrict the number of varying parameters, the computation fraction is uniformly set to $K = 0.7$ for all jobs that arrive to the multi-cluster. This is not a limitation of the model nor the simulator, but rather an imposed constraint for the sake of simplicity.

In order to study to impact of communication, the jobs must be characterized by a per processor bandwidth *PPBW*. We chose to hold every job's bisection bandwidth constant for a particular run of the simulator. This produces a varying *PPBW* due to the varying node sizes of jobs within the workload. We calculate the *PPBW* given the bisection bandwidth, *BSBW*, using Equation (8), which is obtained from our model in Equation (1).

$$PPBW_j = BSBW\left(\frac{4\left(N_T^j - 1\right)}{\left(N_T^j\right)^2}\right) \tag{8}$$

Using the parameters specified above, we conducted three distinct simulations, a 2, 4, and 8 cluster simulation. In each simulation, the bisection bandwidth of the job workload is varied over a particular range, where the average job turnaround time and average co-allocated job penalty in the multi-cluster is measured. The penalty is calculated as the ratio of how long the job actually ran to its original execution time.

The BSBW ranges were chosen to show the behavior of the average job turnaround time in the multi-cluster, as it approaches the *No Share* performance. The *First-fit* is used in conjunction with the dynamic communication model developed in this paper to generate the data associated with the *First-fit* curve.

For each iteration of the simulation (i.e. for a particular BSBW), the average penalty co-allocated jobs experience is also calculated. This penalty in then fed into another instance of the simulation using the fixed penalty model, where every job that is co-allocated experiences an increase in its original execution time by a factor equal to the measured co-allocation penalty. The data that was generated by using the fixed penalty model is shown as *Fixed* in each figure.

4.4. Results and observations

The first general observation we make is that in every simulation, the use of the dynamic communication model is not nearly as generous as the fixed penalty model. Although the actual measured co-allocation penalties from the dynamic communication runs are fed

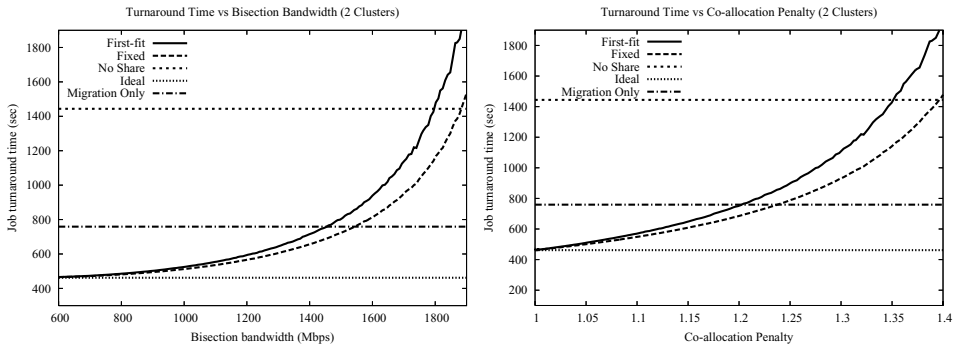

Figure 9. Two cluster case.

directly into the fixed penalty model (i.e. the average co-allocation penalty is identical in both cases), there is a significant difference between the average job turnaround times predicted with each model. The dynamic model accounts for the time varying utilization of the inter-cluster network links, and therefore captures some of the essence of simultaneously co-allocated job interactions in the network.

Additionally, in each scenario (2, 4, 8 clusters: Figures 9–11), the ability to initially migrate jobs *Migration Only* to a remote cluster provides a rather large performance gain over *No Share*. In fact, as the number of clusters increase, so does the performance gain associated with simple job migration. Certainly the gain measured here underestimates the impact associated with activities such as data staging, etc., but we feel that this overhead is relatively small in the multi-cluster context and that it can be accounted for in future work.

By plotting the co-allocation penalty versus turnaround time, it becomes obvious that as the number of clusters increases, the average penalty (using both the dynamic and fixed communication models) that co-allocated job may experience decreases from a range of 1.2 to 1.25 in the two-cluster case, to a range of 1.13 to 1.2 in the eight-cluster case, when

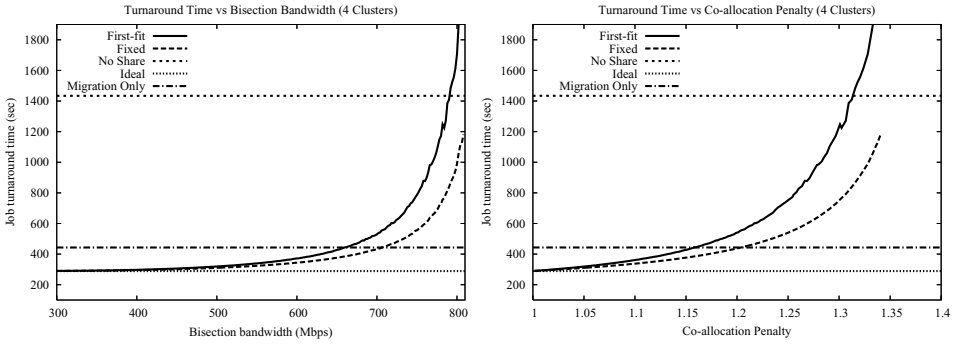

Figure 10. Four cluster case.

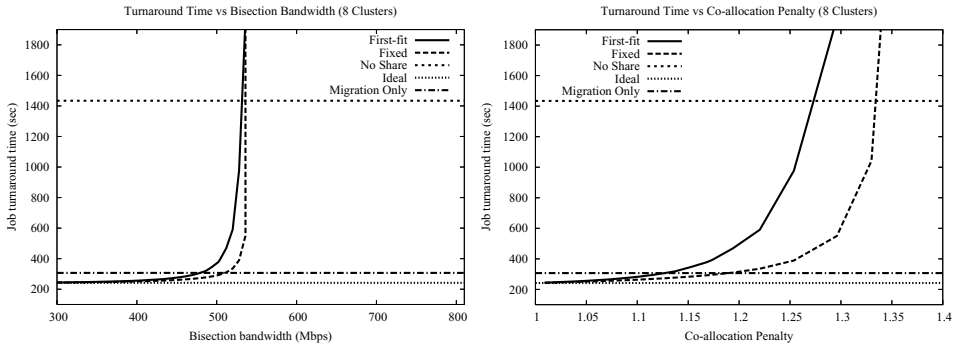

Figure 11. Eight cluster case.

compared to the *Migration Only* strategy. Additionally, when compared to the *No Share* strategy, the acceptable co-allocation penalty also decreases, from a range of 1.35 to 1.4 in the two-cluster case, to a range of 1.25 to 1.35 in the eight-cluster case.

Note that the performance of the first-fit strategy suffers greatly due to the slowdown associated with inter-cluster network saturation, and quickly becomes less effective than simple job migration with no co-allocation.

5. Bandwidth-aware meta-scheduling

In this section we describe several scheduling techniques that attempt to mitigate the impact of simultaneously co-allocated jobs due to inter-cluster network saturation. The algorithms described here will be compared and contrasted in Section 6.

Each of these meta-schedulers consists of a series of modules applied in a given order. Each scheduling module attempts to allocate cluster resources to the given job candidate. These modules are placed in a control loop that sequences the modules, and handles the traversal of the global waiting job queue. This control loop traverses the waiting job queue from head to tail looking for the first job that can make use of available resources.

Each meta-scheduling algorithm has three allocation steps. Each policy attempts to allocate nodes to a given job in the following order: local, migration, co-allocation. The scheduling iteration is formalized by the following template:

Step 1: *Module FCFS*—While not at end of queue, continue, else GOTO Step 5.
Step 2: *Local Allocation*—Apply module LA, if successful then GOTO Step 1, else, continue.
Step 3: *Migration*—Apply module MIG, if successful GOTO Step 1, else, continue.
Step 4: *Co-allocate*—Apply a given co-allocation module. GOTO Step 1.
Step 5: *Termination*

Each module can be classified into three primary categories: 1. a mini-scheduler that *does* have a priori knowledge of a job's Bisection bandwidth (BSBW) (*Class "A"*), 2. a

mini-scheduler that *does not* have a priori knowledge of a job's BSBW (*Class "B"*), and 3. a helper module. In the following paragraphs, each module is described in detail. Note that the class "A" modules will *NEVER* saturate any inter-cluster network link beyond a configurable amount. These modules achieve this ability by knowing a priori how much bandwidth will be used by placing a given number of nodes on a cluster during job co-allocation. Class "B" modules on the other hand, attempt to minimize the level of saturation a given link will experience, but they can not guarantee that a link will not become "over-saturated", since it does not have access to the job's communication characterization. Although class "A" modules may not be practical in general due to their need to possess a priori knowledge of a job's communication characterization, they provide insight into the relative performance of class "B" algorithms.

Helper modules

Module FCFS—module sequencer. This module traverses the global waiting job queue in **F**irst-**C**ome-**F**irst-**S**erved (FCFS) order. It returns the next waiting job to the sequence of modules that represent the meta-scheduling algorithm.

Module LA—-local allocation. This module attempts to allocate a given job locally, by only making use of node resources belonging to the cluster to which it originally arrived.

Module MIG—job migration. This module attempts to migrate a job in its entirety (i.e. no co-allocation) to the cluster with the fewest number of free nodes that can still satisfy the job's resource requirement.

Class "A" modules

Module A1—satisfy. The first scheduling approach we explore ensures that no inter-cluster saturation occurs during the co-allocation phase. In order to map jobs onto the multi-cluster in such a way that completely prevents the slowdown associated with over-saturated inter-cluster network links, it is necessary to first determine the range of nodes that a job j could potentially acquire on link i as a function of the job's bandwidth characterization, as well as the available bandwidth. By letting the left-hand-side of Equation (1) be equal to the available bandwidth on link i, BW_i^{avail}, and then solving the quadratic for n_i^j, Equation (9) is obtained.

$$n_{(1,2)}^{(i,j)} = \frac{1}{2}\left(n_T^j \mp \sqrt{\left(n_T^j\right)^2 - \frac{4BW_i^{\text{avail}}\left(N_T^j - 1\right)}{PPBW_j}}\right) \tag{9}$$

The darkened regions depicted in Figure 12 indicate the potential range of nodes that job j could acquire on link i without over-saturation. Formally, the initial interval of candidate nodes is given by the union of the two regions defined by Equation (9). This interval is then modified to take into account the actual number of nodes, n_i^{avail}, that are presently available on cluster i. The resulting interval, $S_1^{(i,j)}$, is given by Equation (10). This set includes the node ranges defined by the union of the intervals depicted in Figure 12 constrained by the

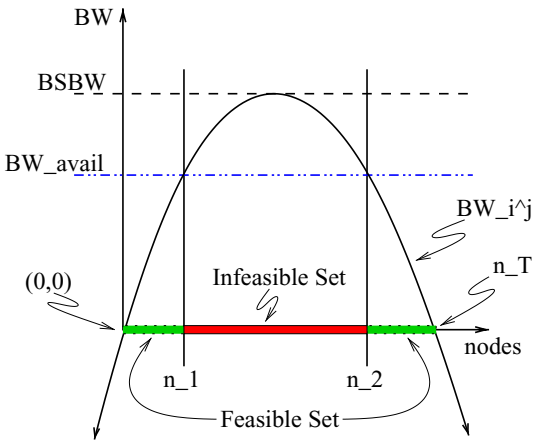

Figure 12. Feasible node ranges.

actual number of free nodes on a given cluster.

$$S_1^{(i,j)} = \left([0, \lfloor n_1 \rfloor] \bigcup \left[\lceil n_2 \rceil, n_T^j\right]\right) \bigcap \left[0, n_i^{\text{avail}}\right] \tag{10}$$

Calculating these intervals for each link results in a set of constraints that must be simultaneously satisfied in order to determine if there exists at least one feasible mapping solution. Collectively the set of constraints is formalized by Equation (11),

$$X_i^j \in S_1^{(i,j)}, \quad \sum_{i=1}^{N} X_i^j = n_T^j \tag{11}$$

where the X_i^j's represent the number of nodes mapped from cluster i for a given job j. This type of system is typically recognized to be an integer constraint satisfaction problem. In order to solve this system, we employ a branch-and-bound brute-force technique that can be configured to either find the first solution that meets all constraints, or to find every solution, depending on whether an objective function is to be applied to find the solution that best meets an optimization criterion. In our experiments for this paper, we have elected to return from this module once the first solution has been identified.

This technique requires the foreknowledge of a job's bandwidth characterization in order to determine the number of nodes that can be placed on a given link during co-allocation. This type of information may not be available a priori. Consequently, developing additional algorithms that do not require this information, yet provide comparable performance, is useful from a practical standpoint.

Class "B" modules

Each of the class "B" modules first identifies all clusters that have links that are saturated beyond a configurable threshold. It then discounts each of these as potential candidates for job co-allocation. These modules will continue to utilize node resources on a given cluster for co-allocation while its network link remains unsaturated. As soon as saturation occurs on a particular network link, this algorithm will then discount its respective cluster for job co-allocation. This implies that a link can only be over-saturated to the extent due to a single job's bandwidth utilization. After completing these two steps, each continues as described below.

Module B1—largest free nodes first. This module sorts the remaining clusters in order of available nodes, and co-allocates the given job starting with the cluster with the largest number of free nodes, and proceeds in order from there.

Module B2—least saturated link first. This module sorts the remaining clusters in order of link saturation, and co-allocates the given job starting with the cluster with the least saturated link, and proceeds in order from there.

Module B3—chunking big-small. This module attempts to co-allocate a "large chunk" (e.g. 75% of node requirement) onto a single cluster. If successful, it will then place the remaining nodes of the job on the remaining clusters, starting with the cluster with the largest number of free nodes, else the module returns unsuccessfully. This module is distinguished from B1 in that it will only successfully schedule a job provided that a "large" partition will fit on a single cluster, whereas B1 will always schedule a job provided that there are enough free multi-cluster resources, regardless of partition sizes. This module attempts to capitalize on two primary observations. Since jobs produce all-to-all communication patterns, the individual bandwidth requirements during co-allocation are minimized when a job is partitioned into a few pieces, one large and perhaps a few small ones. This is in contrast to bisecting the job which results in the maximum bandwidth requirement (Figure 12). However, it may not always be possible to co-allocate a job by partitioning it into at least one "large" piece. In that event, this module simply returns unsuccessfully.

Module B4—load-balancing. This module attempts to co-allocate the job as evenly as possible across the remaining clusters, one node at a time in round-robin fashion.

6. Simulation

In this section, we provide details of our study of bandwidth-aware meta-scheduling algorithms under a variety of algorithmic parameterizations. Figure 13 compares the first-fit strategy described in Section 4 to two bandwidth-aware strategies described in the Section 5. Note that the performance of the first-fit strategy suffers greatly due to the slowdown associated with inter-cluster network saturation, and quickly becomes less effective than simple job migration with no co-allocation, while two of our newest algorithms (A1 & B1) outperform the first-fit strategy over the entire range of tested values. This effect is entirely due to the fact that the first-fit strategy ignores the state of the shared inter-cluster network links.

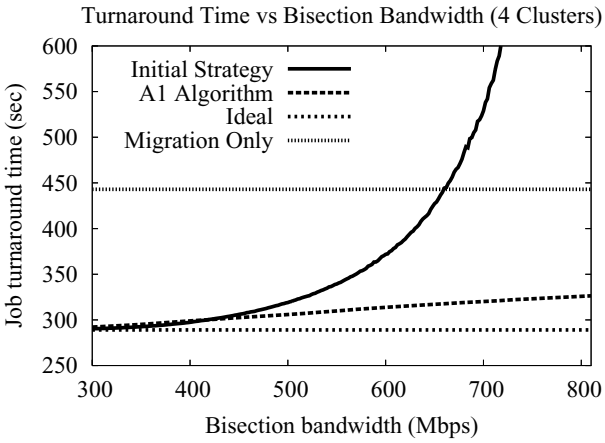

Figure 13. Comparison to bandwidth-aware schedulers.

6.1. Experimental parameters

This subsection describes the set of experimental parameters used in each of the simulations that follow. When comparing our dynamic communication model with the fixed penalty model, we ran simulations on multi-clusters sized at 2, 4, and 8 clusters, however we found that the general trends exhibited in our experiments tended to simply be exacerbated as the number of clusters increased, therefore we focus on multi-clusters containing 4 clusters for the sake of brevity.

Each of the cluster consists of 100 homogeneous computational nodes and has a 1000 Mbps inter-cluster network link to the central switch. The workload presented to each cluster consists of 400,000 jobs. The number of nodes each job requires is taken from a uniform distribution *UNIF*[10, 50] (nodes). The job arrival process is Poisson with the inter-arrival time taken from an exponential distribution with parameter 150 (sec). The base execution time of each job is taken from an exponential distribution with parameter 450 (sec). For simplicity, the computation fraction is uniformly set to $K = 0.7$ for all jobs that arrive to the multi-cluster. Note that these parameters are slightly different from the those used in the comparison study.

6.2. Experimental setup

As with the comparative study (Section 4), each of the scheduling algorithms in this section use the modified FCFS job selection policy (FPFS) outlined in Section 4.1. Additionally, we establish an upper and lower bound for job turnaround time: *Migration Only* and *Ideal*, as described in Section 4.2. These boundaries were established for the multi-cluster characterization used by all of the experiments presented in this section. They are displayed in Figures 22, 24, and 25. In each of of our experiments, two dimensions are explored,

specifically the impact on job turnaround time due to both job BSBW as well as the target inter-cluster link saturation level threshold (LSLT) parameter. In the case of algorithm A1, this percentage is used to drive the simulation during the calculation of the potential number of nodes available for co-allocation (Section 5). For class "B" strategies, this parameter represents the threshold whereby a given cluster's free nodes are discounted as being potential candidates for job co-allocation due to link saturation.

The BSBW parameter for each experiment is swept across a range of 200–900 Mbps. This range was chosen because it represents an interesting range that causes the network to be considerably stressed, and thus allows us to compare and contrast bandwidth-centric co-allocating scheduling algorithms. Additionally, in each experiment the LSLT is swept across a range from 40% to 120% in order to observe the ability of each algorithm to achieve the target LSLT.

6.3. Results and observations

Figures 14–21 show the performance of algorithms A1, B1, B2, B3, and B4. (Note that each scheduling algorithm is named after it's co-allocation module.) On each graph, the x-axis represents the workload's BSBW characterization specified in Mbps. The y axis represents the LSLT specified in percent of total link bandwidth. Note that the LSLT is a parameter provided to the scheduling modules, not a measured value. Finally, the z axis represents the average job turnaround time (TAT). In order to further compare the scheduling algorithms, Figures 22–27 address situations of particular interest.

Figures 22 and 23 show the average job turnaround time as a function of job BSBW while the LSLT is held fixed at 100%. Figures 24, 25, 27, and 26 show the average job turnaround time as a function of the LSLT while the job BSBW's are held fixed at around 300 and 800 Mbps, respectively, representing both low and high levels of job BSBW. The 2D graphs have each been plotted from the same data-sets as the 3D graphs. The Migration Only and

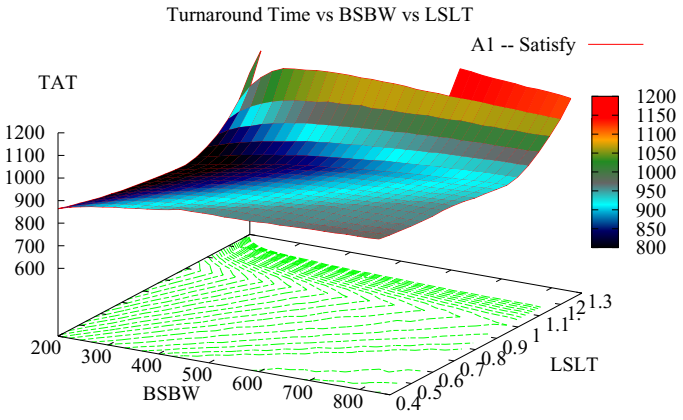

Figure 14. Algorithm A1–Satisfy.

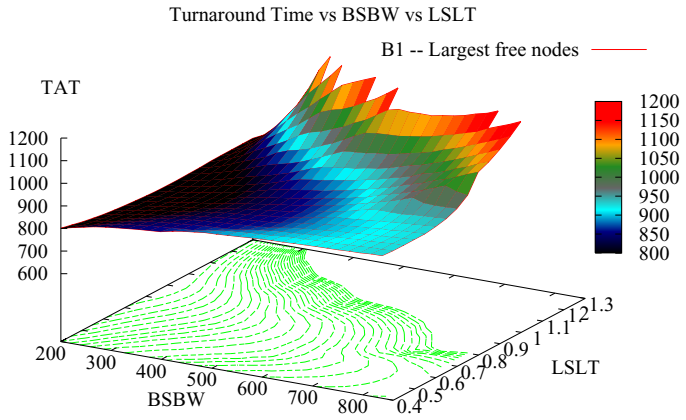

Figure 15. Algorithm B1—Largest free.

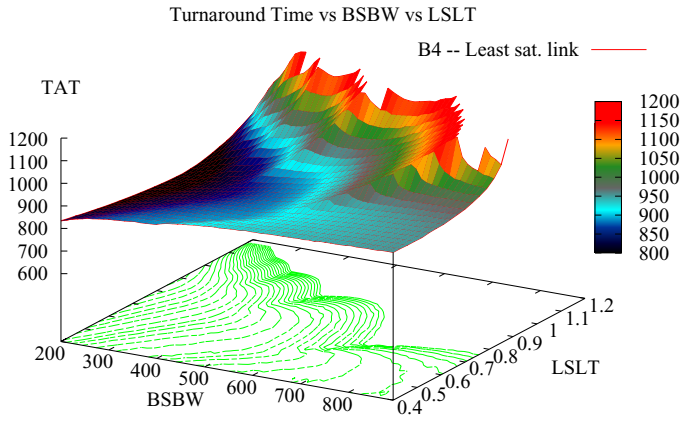

Figure 16. Algorithm B2–Least sat. link.

Ideal schemes have been included in order to compare the performance of the proposed co-allocating algorithms. In particular, a migration-only strategy results in an average job turnaround time of 1087, whereas in the ideal case of unlimited inter-cluster bandwidth, the average job turnaround time is 735.

Since algorithm A1 makes use of a job's communication characterization as well as an integer constraint satisfaction algorithm to determine a job mapping during the co-allocation phase, it can guarantee that a link will never become more saturated than the given LSLT due to job co-allocation. Unfortunately, the calculations involved in A1 are significant and they also require accurate communication characterization of each job. Class "B" algorithms on the other hand, can only guarantee that once a link becomes "over-saturated", it will not become further saturated due to co-allocation. When considering A1's performance, recall

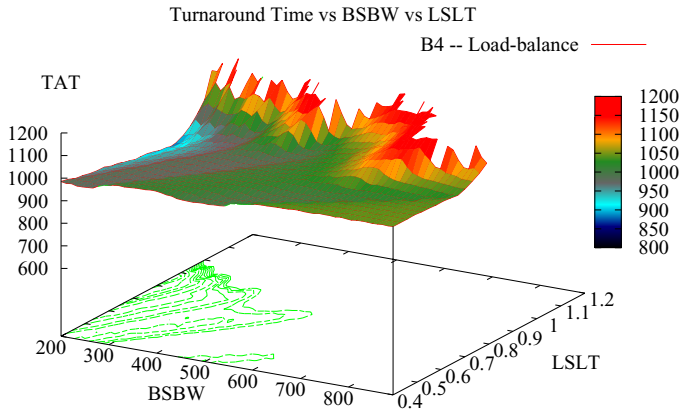

Figure 17. Algorithm B4—Load-balancing.

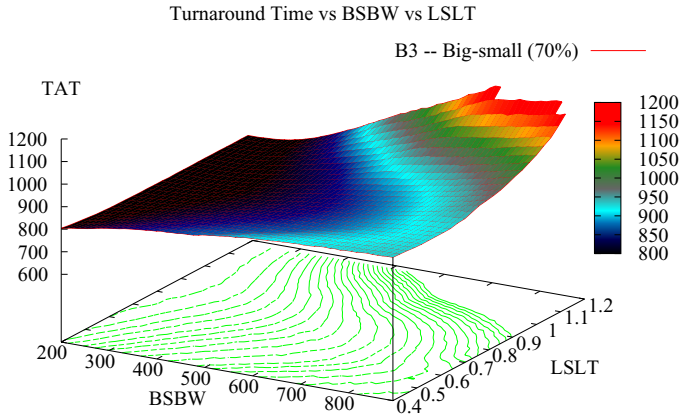

Figure 18. Algorithm B3—Big-small chunk (70%).

that although it can be configured to hold the saturation level of the inter-cluster links at a specified percentage (as seen in the related figures), it would typically be run at 100% saturation level for maximum performance. A1 has been run across the range of saturation levels as a means by which to illustrate the difference between an algorithm (A1) that can guarantee that a link will never be more saturated than a given threshold, versus the class B algorithms that can only *attempt* to limit the saturation level.

Each of the class "B" algorithms can be compared to A1 in order to determine how close they come to approximating it's behavior. Algorithm A1 has a very well-defined and stable response to changes in job BSBW. Although A1 can guarantee that an inter-cluster network link will never become over-saturated as a result of a job co-allocation, this does not imply that it will always produce the best overall performance. In particular, a slight

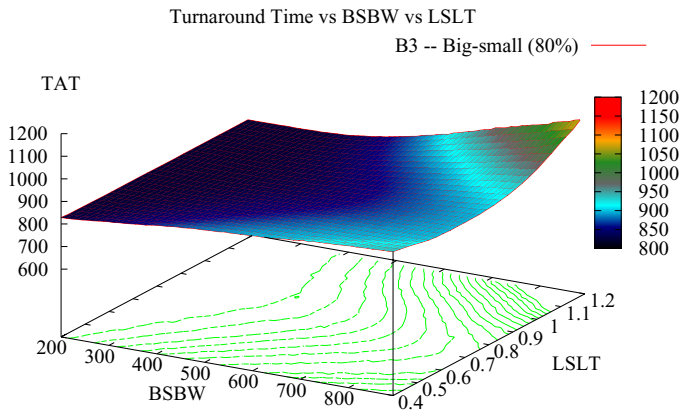

Figure 19. Algorithm B3–Big-small chunk (80%).

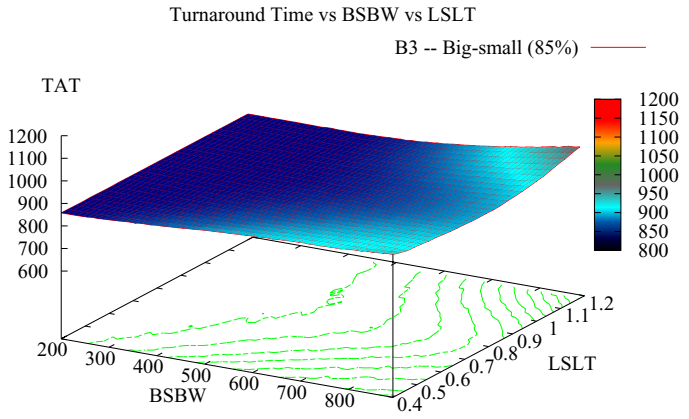

Figure 20. Algorithm B3—Big-small chunk (85%).

over-saturation of network links can in fact be beneficial. This is especially the case when the average waiting time in the queue can be reduced by an amount that exceeds the average increase in job execution time due to the over-saturation. In these cases, job execution slowdown due to inter-cluster network utilization is offset by the fact that more jobs are run earlier due to co-allocation. Therefore, there is a sufficient reduction in queue time that ultimately results in better overall performance in average job turnaround time.

The most interesting of the class "B" algorithms is B3 (big-small chunk). Figures 18–21 show the performance of the B3 algorithm B3 when the chunk size threshold is set to 70, 80, 85, and 90 percent respectively. Algorithm B3 is extremely stable with respect to variation in job BSBW. For the sake of clarity, Figures 23, 26, and 27 have been included to contrast the performance of B3 with respect to chunk size threshold. Note that the B3

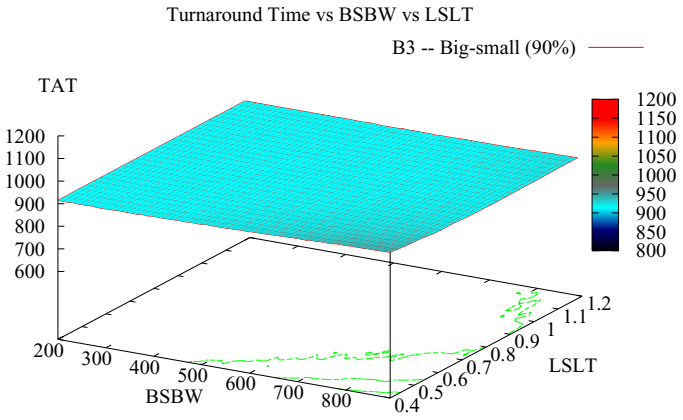

Figure 21. Algorithm B3—Big-small chunk (90%).

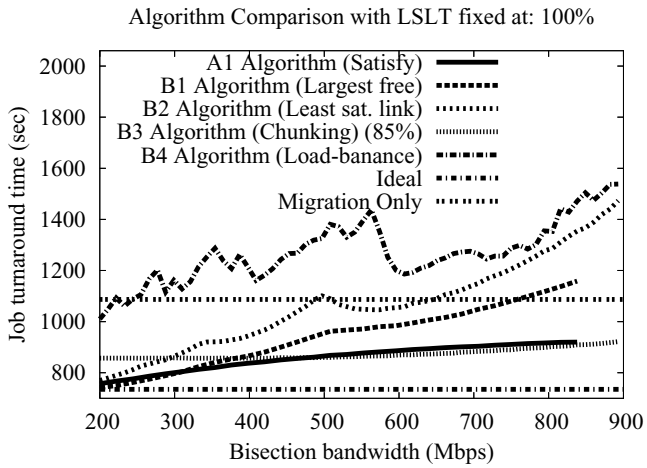

Figure 22. Comparison at 100% saturation.

algorithm outperforms A1 in a variety of circumstances. This is due to the fact that B3 trades over-saturation of inter-cluster network resources for decreased waiting time in the queue. Indeed, even when the LSLT is set to 100%, the B3 algorithms exhibit better performance than A1. Note that as the chunk size approaches 100% (i.e. the entire job) the performance of B3 approaches that of Migration Only; a reassuring result. Additionally, as the chunk size decreases, the performance approaches that of B1, since both co-allocate starting with the largest partition possible. The difference is that B3 will only co-allocate a job when a large portion can fit on a single cluster, whereas B1 will *always* co-allocate a job provided that there are sufficient total resources.

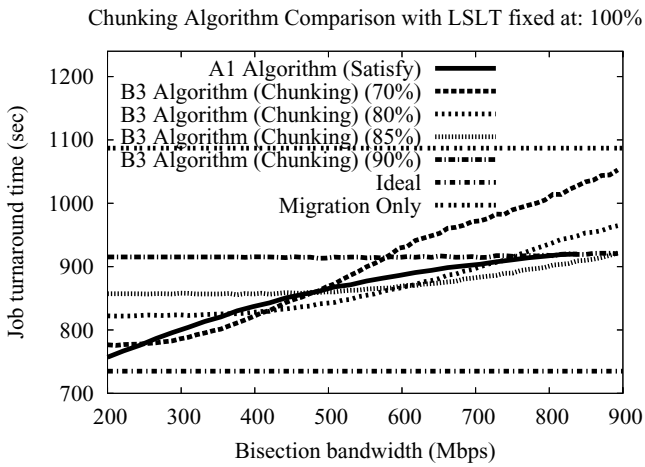

Figure 23. B3 Comparison at 100% saturation.

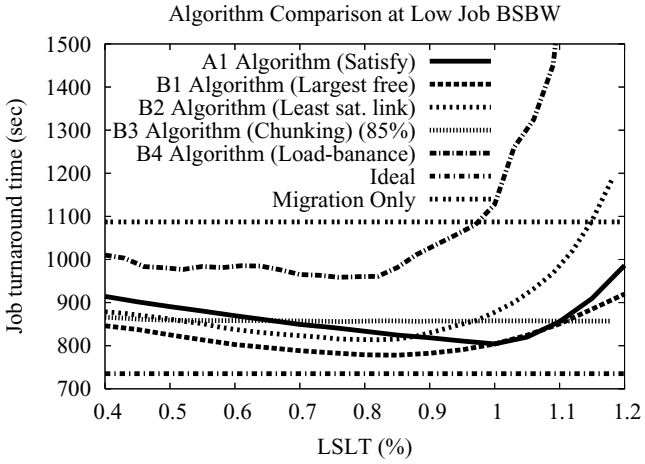

Figure 24. Comparison at low job BSBW.

It is worth noting that the B3 algorithm provides the best overall performance compared to A1. Additionally, it is considerably more stable than the other algorithms in class "B", with respect to variation in job BSBW. It is also worth noting that algorithm B4 (load-balancing) provides the worst overall performance. This is due to the fact that B4 co-allocates a job by spreading it as evenly as possible across the available nodes resources, and in doing so, consumes a substantial fraction of available inter-cluster network bandwidth.

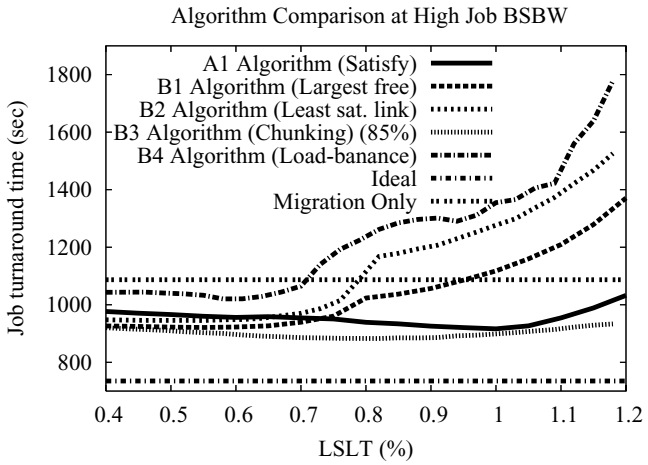

Figure 25. Comparison at high job BSBW.

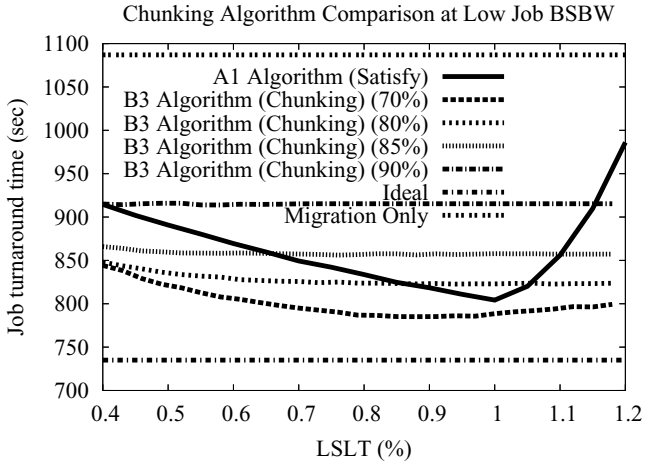

Figure 26. B3 comparison at low job BSBW.

6.4. Scheduler complexity

In addition to evaluating each meta-scheduling algorithm's ability to effectively map jobs onto the multi-cluster, we also wish to provide a brief analysis of their computational complexities. Each of the meta-scheduling algorithms described in this paper consists of a module where the bulk of the decision-making computation is situated. In order to compare the cost of evaluating each algorithm, Table 1 has been provided. (Note: n is the number of jobs waiting in the global job queue, p is the number of nodes required by the given job, and m is the number of clusters.)

Table 1. Algorithm run-time analysis

Module	Complexity	Time (μ Sec)
A1	$O(n \cdot p^m)$	69.6
B1	$O(m \cdot \log(m) + n)$	0.98
B2	$O(m \cdot \log(m) + n)$	0.89
B3	$O(m \cdot \log(m) + n)$	1.28
B4	$O(p)$	1.10

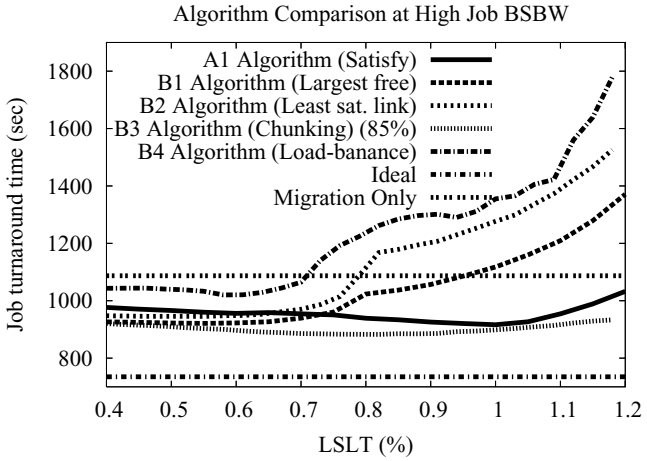

Figure 27. B3 comparison at high job BSBW.

"Complexity" is the standard order-complexity associated with each of the core of meta-scheduling modules. "Time", calculated from data provided by gprof, is the average amount of time spent making a co-allocation scheduling decision by the C code implementation used in the simulations described in this section. While this time may not represent the *total* time in an actual scheduler, it is nevertheless indicative of the scheduling *computation* time.

In the case of module A1, for each of the n jobs waiting in the queue, a brute-force integer constraint satisfaction problem solver is run to determine if the given job is a valid candidate for co-allocation. The kernel of this solver consists for a set of m nested for-loops, each of which iterates from $0 \rightarrow p$, resulting in the p^m factor.

In modules B1, B2, and B3, the $m \cdot \log(m)$ term comes from sorting the list of m clusters using quick-sort. After the sort completes, the global job queue must be traversed in FCFS order to locate a job for co-allocation, thus resulting in the n term.

In module B4, the p term comes from the round-robin traversal of the list of available clusters in an attempt to allocate all p nodes of the given job.

7. Conclusions

In this paper, we examine job scheduling on computational multi-clusters, an important emerging class of "grid-like" architectures. As multi-cluster systems become more prevalent, techniques for efficiently exploiting these resources become increasingly significant.

A critical aspect of exploiting multi-cluster resources is the challenge of scheduling. Previous studies in algorithms for multi-cluster scheduling use a fixed-penalty model for scheduling across cluster boundaries. In this work, we show the potential shortcomings of a fixed-penalty model, and present an alternative bandwidth-centric job communication model that is capable of taking into account time-varying network utilization as a means by which to capture the interaction and impact of simultaneously co-allocating jobs across multiple clusters. We find that the fixed-penalty model is more generous in its prediction of job turnaround time than is our dynamic communication model. Additionally, we see that the penalty that co-allocated jobs can experience without causing a severe performance impact decreases as the number of clusters increase, especially when the meta-scheduler ignores shared network resource usage.

Additionally, we present several bandwidth-aware co-allocating meta-schedulers that take into account inter-cluster network utilization as a means by which to mitigate the slowdown associated with the interaction of simultaneously co-allocated jobs in a dedicated computational multi-cluster. We make use of our bandwidth-centric parallel job communication model to capture the time-varying utilization of shared inter-cluster network resources. By doing so, we are able to evaluate the performance of multi-cluster scheduling algorithms that focus not only on node resource allocation, but also on shared inter-cluster network bandwidth.

We find that it is challenging to design a scheduling algorithm that does not have a priori knowledge of a job's communication characterization, and yet provides comparable performance to one that does. We implemented a variety of such algorithms, and found that co-allocating jobs when it is possible to allocate a large fraction (85%) on a single cluster provides the best performance in mitigating the impact that co-allocated jobs experience due to the slowdown caused by inter-cluster network saturation.

8. Future work

The work presented in this paper attempts to address the design and evaluation of bandwidth-aware scheduling algorithms for mapping jobs across multiple clusters. In doing so, we have made several assumptions related to both the multi-cluster architecture as well as the parallel jobs that execute on these platforms. We have assumed that all node resources are homogeneous and that there is only one processor per node. We have also assumed that the parallel jobs are characterized by global all-to-all communication phases. Under these assumptions, we are able to establish a model that describes the impact that network saturation has on the ultimate runtime of a given job. By making use of this model, we can then evaluate the effectiveness of scheduling algorithms that aim to mitigate this impact.

In order to provide more realism, we intend to relax the constraints mentioned above; however, in doing so, we must extend the models to capture the impact that resource

heterogeneity has on the runtime performance of varying classes of parallel jobs. We would need to address such issues as the structure and frequency of communication synchronizations. We would also need to explore both static and dynamic application load balancing in order to simulate the behavior of such jobs in the presence of heterogeneous computational resources. As a result, new criteria would need to be considered in making scheduling decisions.

The relaxation of these constraints would afford us the opportunity to further refine the scheduling algorithms to take into account both a growing number of specific architectural features as well as parallel program attributes.

Acknowledgments

This work was supported in part by the ERC Program of the National Science Foundation under Award Number EEC-9731680. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the National Science Foundation. Special thanks to Phil Carns, Nathan DeBardeleben, and Mike Speth.

References

1. A. I. D. Bucar and D. H. J. Epema. The influence of communication on the performance of co-allocation. In *7th Workshop on Job Scheduling Strategies for Parallel Processing*, in conjunction with ACM Sigmetrics 2001, pp. 66–86, June 2001.
2. A. I. D. Bucar and D. H. J. Epema. The performance of processor co-allocation in multicluster systems. In *3rd International Symposium on Cluster Computing and the Grid*, pp. 302–309, May 2003.
3. C. Ernemann, V. Hamscher, A. Streit, and R. Yahyapour. Enhanced algorithms for multi-site scheduling. In *Grid Computing—GRID 2002, Third International Workshop, Baltimore, MD, USA, November 18, 2002, Proceedings*, pp. 219–231, 2002.
4. C. Ernemann, V. Hamscher, A. Streit, R. Yahyapour, and U. Schwiegelshohn. On adgantages of grid computing for parallel job scheduling. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02) Berlin Germany May 21*, pp. 31–38, 2002.
5. D. G. Feitelson. Metrics for parallel job scheduling and their convergence. In *Job Scheduling Strategies for Parallel Processing*, vol. 2221, pp. 188–206, 2001.
6. D. Jackson, Q. Snell, and M. Clement. Core algorithms of the maui scheduler. In *7th Workshop on Job Scheduling Strategies for Parallel Processing*. In conjunction with ACM Sigmetrics 2001, June 2001.
7. W. M. Jones, L. W. Pang, D. Stanzione, and W. B. Ligon III. Bandwidth-aware co-allocating meta-schedulers for mini-grid architectures. In *Proc. of the IEEE International Conference on Cluster Computing*, September 2004.
8. W. M. Jones, L. W. Pang, D. Stanzione, and W. B. Ligon III. Job communication characterization and its impact on meta-scheduling co-allocated jobs in a mini-grid. In *Proc. of the IEEE 18th International Parallel and Distributed Processing Symposium: Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems*, April 2004.
9. D. Lifka. The ANL/IBM SP scheduling systems. In *Proc. of the 1st Workshop on Job Scheduling Strategies for Parallel Processing*, vol. 949, pp. 295–303. LNCS, 1995.
10. J. Sinaga, H. Mohamed, and D. H. J. Epema. A dynamic co-allocation service in multicluster systems. In *10th Workshop on Job Scheduling Strategies for Parallel Processing (in conjunction with Sigmetrics-Performance 2004)*, New York, June 2004.

11. S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan. Characterization of backfilling strategies for parallel job scheduling. In *IEEE International Conference on Parallel Processing Workshops*, pp. 514–519, August 2002.
12. Y. Zhang, H. Franke, J. Moreira, and A. Sivasubramaniam. An integrated approach to parallel scheduling using gang-scheduling, backfilling, and migration. In *IEEE Transactions On Parallel and Distributed Systems*, vol. 14, pp. 236–247, March 2003.