Bandwidth-aware Co-allocating Meta-schedulers for Mini-grid Architectures *

William M. Jones† 864-656-7367 wjones@parl.clemson.edu
Louis W. Pang† 864-656-7367 plouis@parl.clemson.edu
Dan Stanzione‡ 703-292-8121 dstanzio@nsf.gov
Walter B. Ligon III† 864-656-1224 walt@parl.clemson.edu

†Parallel Architecture Research Lab
Department of Electrical and Computer Engineering
Clemson University
105 Riggs Hall
Clemson, SC 29634-0915
http://www.parl.clemson.edu

‡National Science Foundation
Division of Graduate Education
4201 Wilson Blvd Suite 907
Arlington, VA 22230

Abstract

The interaction of simultaneously co-allocated jobs can often create contention in the network infrastructure of a dedicated computational grid. This contention can lead to degraded job run-time performance. In this paper, we present several bandwidth-aware co-allocating meta-schedulers. These schedulers take into account inter-cluster network utilization as a means by which to mitigate this impact. 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 grid scheduling algorithms that focus not only on node resource allocation, but also on shared inter-cluster network bandwidth.

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 mini-grid.

A mini-grid is distinguished from a traditional computational grid in that the mini-grid utilizes a dedicated interconnection network between grid 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*. In this paper, we develop 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 mini-grid. By employing 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 grid scheduling algorithms that focus not only on computational resource allocation, but also on shared inter-cluster network bandwidth. The main focus of this paper is to provide an in-depth explanation of our bandwidth-aware co-allocation algorithms and to analyze their performance characteristics.

In our previous work [5], we developed a communication model that is sensitive to the time-varying contention for bandwidth in the inter-cluster communication links and can be used to determine the impact on the execution times of co-allocated jobs. Our dynamic communication model contrasts other models that consider the communication cost between clusters to be fixed [1], [2], [3], [4]. We found that schedulers designed to allocate node resources across cluster boundaries resulted in poor overall performance over a wide range of workload characterizations and mini-grid configurations due to the interaction simultaneously co-allocated jobs experience as they contend for inter-cluster network bandwidth. Our current research therefore focuses on a range of algorithms with varying levels of complexity that attempt to mitigate this impact.

*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.

2 The Model

In this section we characterize the parallel job model as well as the mini-grid architecture. We provide a brief explanation of the communication model used, as well as a strategy to account for the time-varying inter-cluster network utilization.

2.1 Mini-grid and Parallel Job Model

We consider the mini-grid 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. Each node in the mini-grid has a single processor and a single network interface card. Jobs can be co-allocated (Figure 2) in a mini-grid by allocating nodes from different clusters to the same job.

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 [7] is employed in mapping the job onto the mini-grid; 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. T_P and T_C therefore form a basis for the best-case execution time of a given job when it is co-allocated in the mini-grid. 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.

2.2 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 amount 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 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.

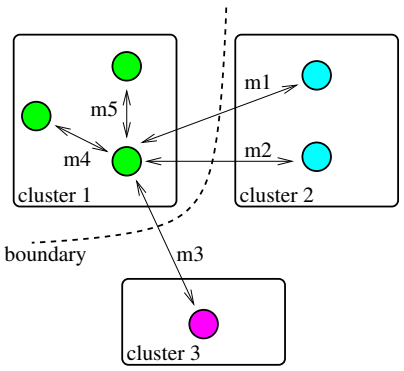

Figure 1. Bandwidth calculation example

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

The first factor in this equation is the total bandwidth required by all the nodes associated with job j on cluster C_i. The second factor represents the fraction of the messages generated by each node on cluster, C_i, that is destined for non-local nodes. For example, suppose that a job consists of six total nodes and has been mapped onto a grid consisting of three clusters, as shown in Figure 1. 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.

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

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

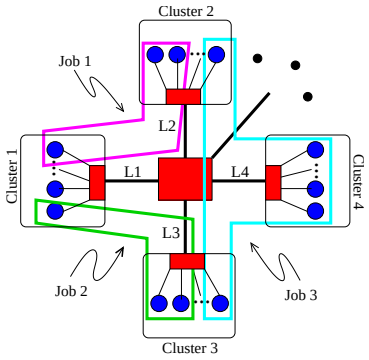

Figure 2. Co-allocation example

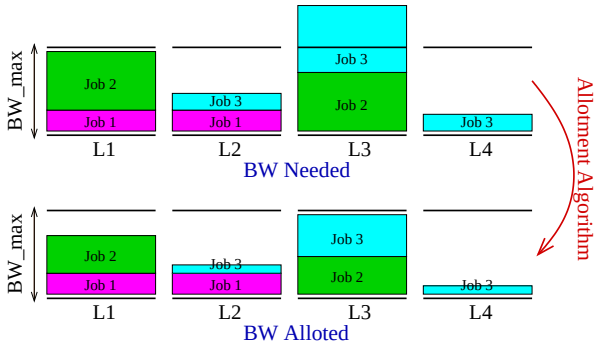

Figure 3. Bandwidth allotment

2.3 Job Slowdown due to Bandwidth Saturation

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: a new job is co-allocated in the mini-grid, or 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

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

where set J_i is the set of all jobs that span link i. If $BW_i^{sat} \geq 1.0$ then link i is *not* saturated; otherwise, if $(0.0 \leq BW_i^{sat} < 1.0)$, 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 a co-allocated job terminates, an algorithm is applied in order to determine the amount of bandwidth ultimately alloted to each job on each link. The amount of bandwidth each job receives is limited by the most saturated link over which it spans. This algorithm is described in detail in our previous work [5] (Figures 2 and 3).

Each affected job's bandwidth allotment on each link over which it spans is reduced in order to accommodate it's most saturated link. Equation 4 formalizes the bandwidth slowdown associated with job j, where link k may be any link over which the job spans.

$$BW_{(k,j)}^{sd} = \frac{BW_{(k,j)}^{alloted}}{BW_k^j} \tag{4}$$

Now that the communication slowdown factor is known, the residual execution time, T_E^R, of a job can be calculated as a function of both the residual communication and computation times (T_C^R and T_P^R 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 5 and 6 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.

$$T_E^R = \overbrace{(T_C^{R'} - T_C^{\Delta})(BW_{(k,j)}^{sd'})(BW_{(k,j)}^{sd})^{-1}}^{T_C^R} + \underbrace{(T_P^{R'} - T_P^{\Delta})}_{T_P^R} \tag{5}$$

where

$$T_P^{\Delta} = \frac{\Delta T}{T_E^{R'}} T_P^{R'} \quad , \quad T_C^{\Delta} = \frac{\Delta T}{T_E^{R'}} T_C^{R'} \tag{6}$$

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 5.

As inter-cluster state changing events occur, the residual times are recalculated based on equations 5 and 6. Due to these recalculations, the job's end-event can slide forward (later) or backward (earlier) in time (Figure 4), reflecting

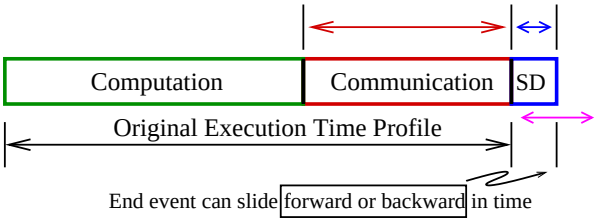

Figure 4. Bandwidth slowdown effect

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 Meta-scheduling Algorithms

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 grid. A meta-scheduler is expected to work in conjunction 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 mini-grid. In order to address the impact of job co-allocation on the performance of the mini-grid, the following strategies and policies are analyzed.

3.1 Previous Strategies and Policies

In our previous work [5], we implemented several algorithms to evaluate our communication model and the impact of co-allocating jobs in a mini-grid. A brief description of one algorithm and two base-line configurations is provided here. A description of our current bandwidth-aware meta-scheduling algorithms is provided in Section 3.2.

Our initial strategy performed 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. We refer to this strategy as **Initial Strategy**.

In order to establish a reasonable upper and lower bound for job turnaround time, two 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

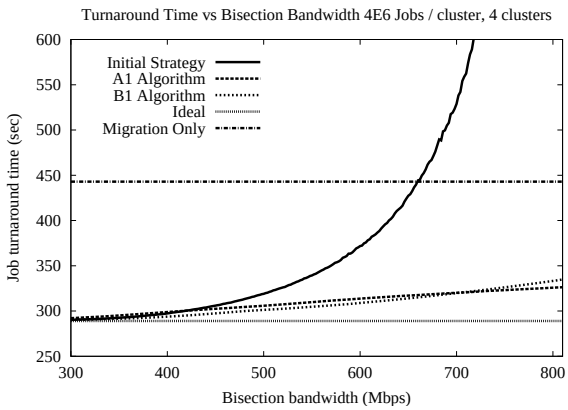

Figure 5. Job Turnaround Time

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.

Both the *Ideal* as well as the *Migration Only* strategies appear as horizontal "limits" in Figure 5, since they are unaffected by a job's bisection bandwidth.

A full description of these algorithms can be found in [5]. The results of one test from our previous work are summarized in Figure 5. Note that the performance of the initial 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) out-perform our initial strategy over the entire range of tested values. These algorithms (and others) are described in detail in the next section.

3.2 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 4.

Each of our meta-schedulers consists of a series of modules applied in a given order. Each scheduling module attempts to allocate grid 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. This contrasts a traditional policy known as *EASY backfilling* [6] that is used in many production grid schedulers, such as Maui.

Since we make use of the bandwidth-centric communication model, only an estimate of a job's end event is known at any instant in time. This is due to the fact that a job's end event can slide forward and backward in time depending on the communication contention in the inter-cluster network links (Figure 4). 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 as a potential policy.

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 they do not have access to the job's communication characterization.

Helper modules

Module FCFS – Module sequencer This module traverses the global 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.

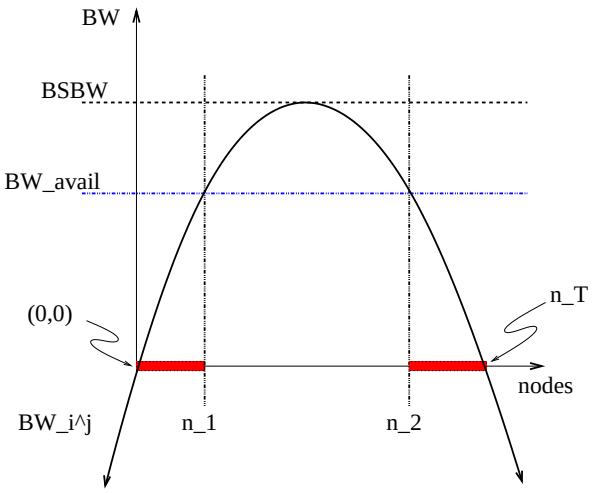

Figure 6. Feasible node ranges

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 mini-grid 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 the quadratic for n_i^j, equation 7 is obtained.

$$n_{(1,2)}^{(i,j)} = \frac{1}{2}\left(n_T^j \mp \sqrt{(n_T^j)^2 - \frac{4BW_i^{avail}(N_T^j-1)}{PPBW_j}}\right) \quad (7)$$

The darkened regions depicted in Figure 6 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 7. This interval is then modified to account for 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 8. This set includes the node ranges defined by the union of the intervals depicted in Figure 6 constrained by the actual number of free nodes on a given cluster.

$$S_1^{(i,j)} = \left([0, \lfloor n_1 \rfloor] \bigcup [\lceil n_2 \rceil, n_T^j]\right) \bigcap [0, n_i^{avail}] \quad (8)$$

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 9,

$$X_i^j \in S_1^{(i,j)}, \quad \sum_{i=1}^{N} X_i^j = n_T^j \tag{9}$$

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 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 grid 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 6). 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.

4 Simulation

Beosim [5] is a discrete event driven simulator designed to model a mini-grid as a collection of computational clusters connected via a dedicated interconnection network. 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 $D_i^{nodes} \sim UNIF[n_1^i, n_2^i]$. The fraction of *the total execution time* that initially represents computation is set to a constant, K_i, for all jobs.

4.1 Experimental Parameters

This subsection describes the set of experimental parameters used in all simulations in this paper. In our previous work [5], we ran simulations on mini-grids 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, in this paper, we focus on mini-grids containing 4 clusters for the sake of brevity.

Each cluster in the mini-grid 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, each with a node requirement 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 mini-grid.

In order to study the 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.

4.2 Experimental Setup

In order to establish an upper and lower bound for the job turnaround time metric, two baseline simulations were conducted to identify these levels: **Migration Only** and **Ideal** (Section 3.1).

These boundaries were established for the mini-grid characterization used by all of the experiments presented in this paper. They are displayed in Figures 13, 15, and 16. 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 3.2). 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.

4.3 Results and Observations

Figures 7 - 12 show the performance of algorithms A1, B1, B2, B3, and B4. (Note that each scheduling algorithm is named after its 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 13 - 18 address situations of particular interest.

Figures 13 and 14 show the average job turnaround time as a function of job BSBW while the LSLT is held fixed at 100%. Figures 15, 16, 17, and 18 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 datasets as the 3D graphs. The Migration Only and 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 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 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 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

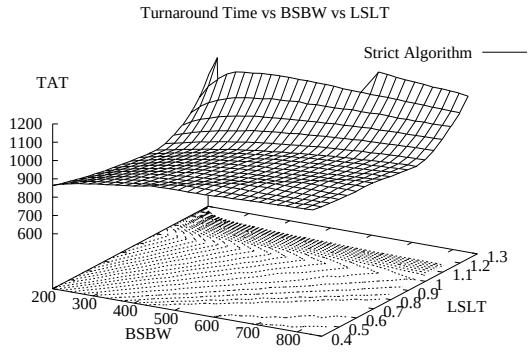

Figure 7. Algorithm A1 – Satisfy

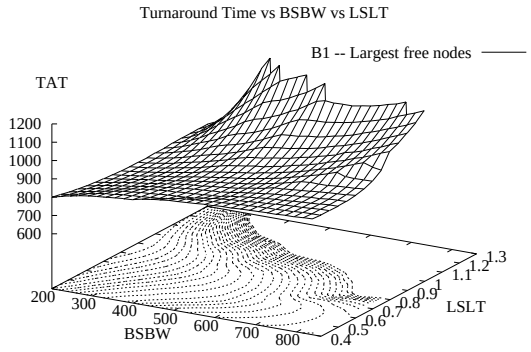

Figure 8. Algorithm B1 – Largest free

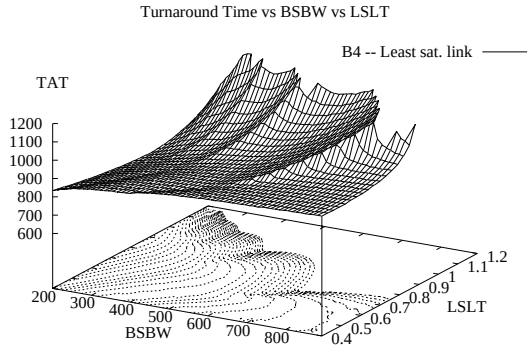

Figure 9. Algorithm B2 – Least sat. link

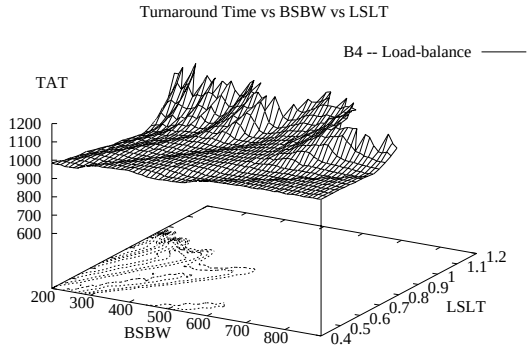

Figure 10. Algorithm B4 – Load-balancing

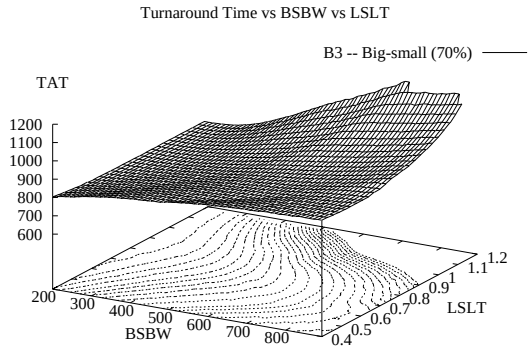

Figure 11. Algorithm B3 – Big-small chunk (70%)

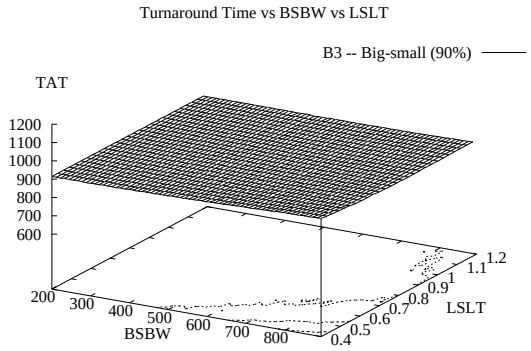

Figure 12. Algorithm B3 – Big-small chunk (90%)

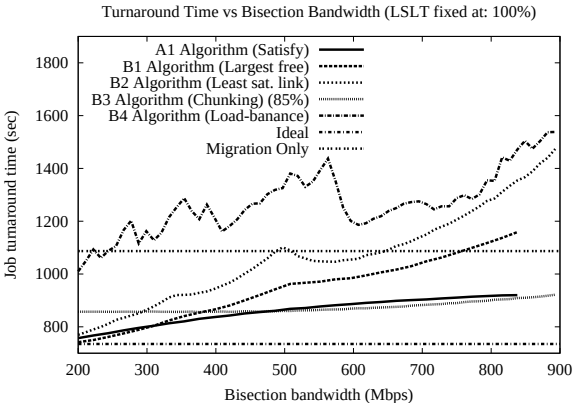

Figure 13. Comparison at 100% Saturation

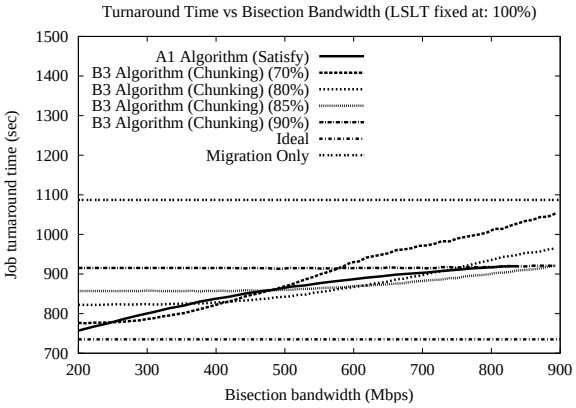

Figure 14. B3 Comparison at 100% Saturation

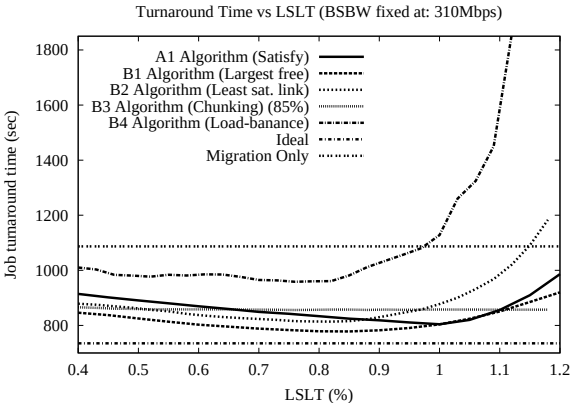

Figure 15. Comparison at Low Job BSBW

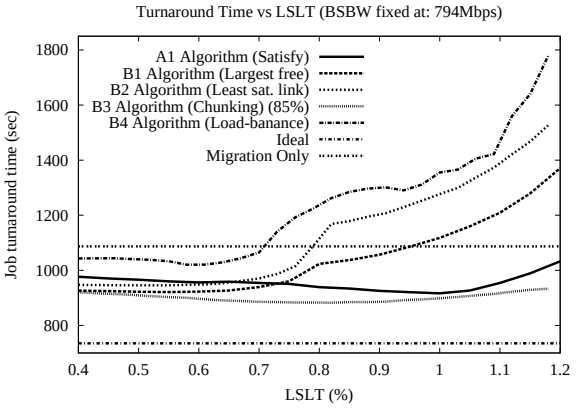

Figure 16. Comparison at High Job BSBW

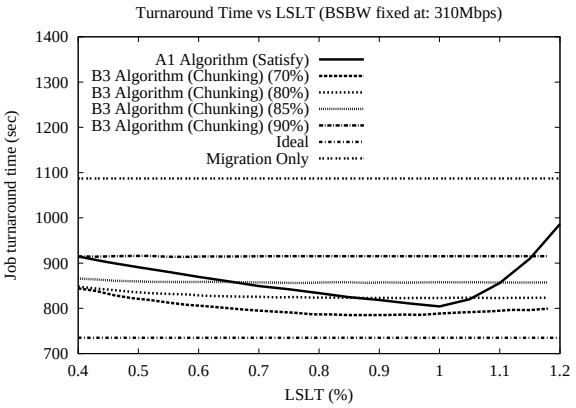

Figure 17. B3 Comparison at Low Job BSBW

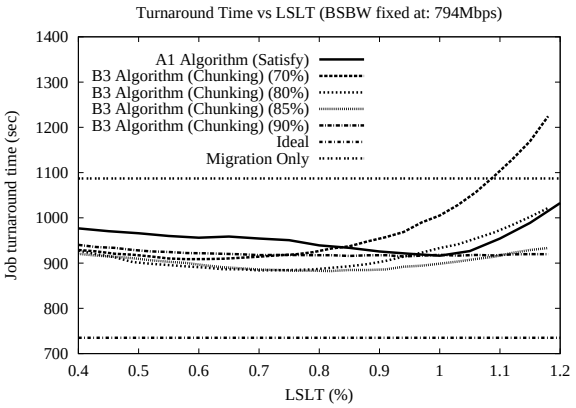

Figure 18. B3 Comparison at High Job BSBW

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 11 and 12 show the performance of the B3 algorithm as the chunk size threshold is increased from 70 to 90 percent respectively. Algorithm B3 is extremely stable with respect to variation in job BSBW. For the sake of clarity, Figures 14, 17, and 18 have been included to contrast the performance of B3 with respect to chunk size threshold. Note that the B3 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.

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.

5 Conclusions and Future Work

In this paper, 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 grid. 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 grid 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.

Our future work will include further development of scheduling algorithms that fall into both classes presented in this paper. Additionally, we have made recent progress in empirically verifying our communication model, and we intend to follow this to it's natural end. We plan to investigate the impact of a workload with a mixture of jobs that have varying bisection bandwidths. We are currently exploring adapting our job communication model to include different types of local and global communication patterns by making use of k-ary n-cubes to model application communication structure.

Acknowledgments: 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*, pages 66–86. in conjunction with ACM Sigmetrics 2001, 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*, pages 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*, pages 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*, pages 31–38, 2002.

[5] 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.

[6] 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*, pages 514–519, August 2002.

[7] 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*, volume 14, pages 236–247, March 2003.