Network-aware Selective Job Checkpoint and Migration to Enhance Co-allocation In Multi-cluster Systems[†]

William M. Jones[*,‡]

Computer Science Department, Coastal Carolina University, Conway, SC 29526
http://www.coastal.edu/cs/

SUMMARY

Multi-site parallel job schedulers can improve average job turn-around time by making use of fragmented node resources available throughout the grid. By mapping jobs across potentially many clusters, jobs that would otherwise wait in the queue for local resources can begin execution much earlier; thereby improving system utilization and reducing average queue waiting time.

Recent research in this area of scheduling leverages user-provided estimates of job communication characteristics to more effectively partition the job across system resources. In this paper, we address the impact of inaccuracies in these estimates on system performance and show that multi-site scheduling techniques benefit from these estimates, even in the presence of considerable inaccuracy. While these results are encouraging, there are instances where these errors result in poor job scheduling decisions that cause network over-subscription. This situation can lead to significantly degraded application performance and turnaround time.

Consequently, we explore the use of job checkpointing, termination, migration, and restart (CTMR) to selectively stop offending jobs to alleviate network congestion and subsequently restart them when (and where) sufficient network resources are available. We then characterize the conditions and the extent to which the process of CTMR improves overall performance. We demonstrate that this technique is beneficial even when the overhead of doing so is costly. Copyright © 2009 John Wiley & Sons, Ltd.

KEY WORDS: parallel job scheduling; checkpointing; migration; clusters; grid scheduling

[*]Correspondence to: W. M. Jones, Department of Computer Science, Coastal Science Center, Coastal Carolina University, Conway, SC

[†]This is an extended version of *"Using Checkpointing to Recover from Poor Multi-site Parallel Job Scheduling Decisions"* that appeared in the 5th International Workshop on Middleware for Grid Computing, held in conjunction with the 2007 ACM/IFIP/USENIX 8th International Middleware Conference, 26-Nov-2007, [1].

[‡]This is the author's version of the work. The definitive version of the work was published 13 Feb. 2009 at http://www3.interscience.wiley.com/journal/122198926/abstract.

INTRODUCTION

As cluster computing becomes more commonplace, industrial and academic research parks often purchase several clusters to meet their computational needs. These geographically co-located clusters can be connected via an interconnection network to form a larger computational grid resource known as a multi-cluster or super-cluster [2]. This configuration allows flexibility for distributing parallel jobs among available clusters; however, it also increases the complexity of managing both computing and networking resources.

As multi-cluster systems become more prevalent, techniques for efficiently exploiting these resources become increasingly significant. A critical aspect of exploiting these systems is the challenge of job scheduling [3], [4]. Intelligent schedulers can make use of information related to job communication structure and inter-cluster bandwidth availability to improve average job response time by selectively mapping parallel jobs across potentially many clusters in a process known as job co-allocation or multi-site scheduling [5, 6, 7] [†].

One of the caveats of this type of resource sharing is that user-provided job communication estimates may be inaccurate. Since multi-site scheduling techniques can explicitly make use of job bandwidth requirements to partition the job across clusters, any inaccuracies could have adverse effects on the overall system performance. In fact, the question becomes one of determining how sensitive the job schedulers actually are to the quality of these estimates. In addressing this question, one can begin to evaluate the trade-off between the cost of providing more accurate information, and the improvement obtained in doing so.

The notion of making use of user estimates for the purpose of job scheduling and exploring the effects of their inaccuracies [8], [9] is not new. If fact, many backfilling production schedulers (Maui/Moab [10] and IBM's LoadLeveler) make use of user-provided estimates of job runtime to determine when and how to backfill jobs. However, in multi-site parallel job scheduling, communication characterizations are used in addition to runtime estimates to allow the scheduler to partition the job across two or more clusters by intelligently managing both node and *network* resources [5]. Specifically, our contributions are centered around exploring system behavior in the presence of inaccurate user-predicted *bandwidth requirements* as opposed to *runtimes*. Understanding the resulting behavior is particularly important in the context of *bandwidth-aware* multi-site parallel job scheduling.

Our initial research has shown that the impact of these inaccuracies ranges from negligible to severe, depending on a number of factors including the relative intensity of inter-process communication [11]. It is therefore equally important to identify mechanisms to mitigate the negative impact of these inaccuracies when they occur. One such technique is to checkpoint an offending job, terminate its current execution, migrate it to a location where more network resources are available, and to subsequently restart its execution, a collective process we refer to as CTMR.

Job checkpointing is a process where the entire state of the application is saved to traditionally persistent storage so that it may be restarted at later time, typically after a

[†]An interesting related effort known as *Dynamic Virtual Clustering* can be found at http://hpc.asu.edu/dvc/.

fatal error [12, 13]. While checkpointing is largely used in parallel and distributed computing to recover from *component failures*, we use it in conjunction with job migration, to recover from scheduling decisions that lead to *network over-subscription*. For example, if a user underestimates a job's bandwidth requirements, the scheduler may co-allocate a job that results in network over-subscription. This in turn causes all jobs that are mapped across over-saturated links to slow down. We focus on job checkpoint, termination, migration, and restart (CTMR) to alleviate network over-subscription and mitigate job slowdown.

In this paper, we begin the analysis by addressing the impact of inaccuracies in user-provided *communication requirement* estimates on overall system performance from several points of view. Furthermore, we demonstrate that multi-site job scheduling techniques benefit from these estimates, even in the presence of considerable inaccuracy. Additionally we quantify the difference in the impact that overestimation causes versus underestimation. We also demonstrate that the extent to which estimate error impacts co-allocation is strongly correlated to the intensity of inter-processor communication.

We then continue the analysis by describing an agent that autonomously decides when to checkpoint, terminate, migrate and subsequently restart jobs to mitigate network over-subscription due to estimate inaccuracies and we provide a rationale behind its parametrization. We subsequently characterize the conditions and the extent to which checkpointing, migration and ultimate the restart improves multi-site parallel job scheduling performance. We demonstrate that checkpointing improves performance even when the overhead of doing so is very costly. Finally we show that at moderate levels of overhead, CTMR can be used to mitigate the negative impact of estimate inaccuracies.

THE MODEL

In this section we describe the parallel job model as well as the multi-cluster architecture. We provide a *very* brief explanation of the communication model used, as well as a strategy to account for the time-varying inter-cluster network utilization. This general technique is based on the work presented in [5]. The interested reader can obtain a detailed treatment of the complete modeling methodology on the BeoSim website [14].

Multi-cluster and Parallel Job Models

As a first step, we consider a multi-cluster to be a collection of arbitrarily-sized clusters with globally homogeneous node. Each cluster has its own internal switch. Additionally, the clusters are connected to one another with a single dedicated link to a central switch. Each node in the multi-cluster has a single 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 to better meet collective needs across the multi-cluster. 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.

A job's execution time, T_E, is a function of two components, the computation time, T_P, and the non-overlapped 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*. They therefore form a basis for the best-case execution time of a given job when it is co-allocated in the multi-cluster. 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.

Communication Characterization

In order to capture both local and global communication characteristics, each job modeled in this paper is assumed to perform both nearest neighbor (2D mesh) and all-to-all personalized communication patterns throughout its execution. Jobs are further characterized by their per-processor bandwidth (PPBW), i.e. the network bandwidth required by each node in the job.

During co-allocation, nodes must communicate across cluster boundaries. This communication requires 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. Each time a new job is co-allocated or a co-allocated job terminates, an algorithm is applied to determine the amount of bandwidth ultimately allotted to each job on each link. The amount of bandwidth each job receives is limited by the most saturated link over which it spans.

As these inter-cluster state changing events occur, the remaining execution and communication times are recalculated based on a number of factors, including available network bandwidth. Due to these recalculations, the job's end-event can slide forward (later) or backward (earlier) in time, reflecting either a degradation or improvement in saturation levels of the inter-cluster links over which it spans. The full description is rather lengthy and can be found described in detail in [5].

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. This is particularly important when considering inaccurate user estimates of bandwidth since they can cause the scheduler to perform co-allocation when sufficient inter-cluster network resources are not available.

JOB CHECKPOINTING AND MIGRATION

Checkpointing is largely used in parallel and distributed computing as a mechanism to recover from failures in system components [15] and has recently been used to improve application and system resilience [16]. As the size of parallel systems increases, the mean time between failures (MTBF) typically decreases [17] . Without checkpointing, a job mapped across a failed component would likely need to be restarted from the beginning, thus resulting in longer turnaround times, and redundant usage of system resources. While this is an extremely important use of checkpointing, it is not the focus of this paper.

Migration is the process where a job is moved from one set of computational and network resources to another. In this case, the job would be migrated to a location where more network resources are available, and if need be, pause the job until sufficient resources become available. By combining the utility of job checkpointing with the flexibility of job migration, we are able to recover from initially poor scheduling decisions by alleviating network congestion, and therefore mitigating job slowdown. In this section, we describe the checkpoint/migration agent and the motivation behind certain decisions regarding its implementation.

Agent Motivation

Determining how and when to checkpoint, terminate, migrate, and restart a job to alleviate network saturation is a difficult question. A naïve approach may simply checkpoint and restart a co-allocated job any time an interconnection network link is over-subscribed. This aggressive approach does not take into account several important factors. In this section we describe a few scenarios that will serve as motivation for the CTMR agent implementation we ultimately use.

First, previous research has shown that occasional over-subscription can actually improve overall job throughput (refer to [5], Section 6.3). For example, the reduction in queue waiting time can outweigh the increase in execution time caused by network saturation. Second, there is the issue of capacity loss. Suppose a co-allocated job were checkpointed to alleviate over-subscription, but there are no jobs waiting in the queue that can immediately make use of the resources freed by the checkpointed job. This is due to two primary reasons: there are no jobs that can fit within the number of nodes freed, or there *are* jobs that could fit; but other scheduling constraints exist prevent job dispatch, such as priorities, backfill reservations, bandwidth requirements, etc. In this case, the decrease in system utilization can lead to a decrease in performance that outweighs the improvement in network saturation. Lastly, it is important to consider the additional time required to checkpoint, terminate, migrate, and restart. If this time is sufficiently long compared to the remaining execution time of the co-allocated jobs, checkpointing can lead to an increase in the jobs' execution times that outweighs the benefit of reducing network over-subscription.

CTMR Agent Implementation

Each of the above factors, among others, plays an important role in the implementation of our checkpoint, termination, migration, and restart (CTMR) agent. The basic agent iteration is as follows:

Step 1: *Identify congestion* - If network saturation exists AND candidate jobs remain, identify the cluster with the most saturated link, L, and proceed; else schedule CTMR agent to run in the future and exit.

Step 2: *Find a candidate job* - Inspect all jobs co-allocated across link L. Find the job with the largest subscription on link L, subject to three constraints: **(1)** the job underestimated its bandwidth requirements, **(2)** the time required to CP (if known) the job is less than some fraction of the job's remaining execution time. *(The determination of this parameter is described below.)*, and **(3)** at least one waiting job could immediately make use of freed resources if candidate job were checkpointed.

Step 3: *Checkpoint job* - If a suitable candidate is found, checkpoint and place at head of local waiting queue; else exit.

Step 4: *Run scheduler* - After checkpoint is complete, run parallel job scheduler to identify and dispatch jobs that can now run using the freed resources.

Step 5: *Re-evaluate system* - Goto step 1.

In addition, a job that initially underestimates its bandwidth requirements is immediately marked as such prior to the first checkpoint / migration. The scheduler then assigns the job a new bandwidth estimate based on observed runtime behavior. If the job causes network over-subscription a second time, it will be checkpointed and restarted again; however it will no longer be considered for job co-allocation. This ensures that its inter-process communications will not traverse any network trunks connecting the participating clusters to the grid. Note, we assume that the communication bottlenecks in such a multi-cluster are in the inter-cluster trunks, i.e., not in the backplane bandwidth of the internal cluster switches.

In Step 2, we make use of a configurable parameter, $CPfrac$, that serves as one factor to determine the suitability of a job as a checkpointing / migration candidate. In particular,$CPfrac$ is used to prevent a job from being checkpointed if the overhead in doing so is “sufficiently” long compared to the remaining execution time. A natural question is how to choose its value. We initially conducted a parameter sweep to determine the value of $CPfrac$ that minimizes average job turnaround time in a number of different scenarios. Specifically, we needed to determine how sensitive this parameter is to other system characteristics. We found that in almost every case we tried, a value of around 4% resulted in the best performance. This was a surprising result and greatly simplified the logic applied in finding a checkpointing candidate. Note, if the time required to checkpoint/restart is not known or unavailable, we assume a generous overhead of 30 minutes.

SIMULATION FRAMEWORK

Using the BeoSim multi-cluster simulator [14], we model each of the four clusters in the grid as a collection of 100 homogeneous nodes connected via an internal switch. The clusters are connected to one another by 1 Gbps network links via a central switch. The workload presented to each cluster consists of 400,000 jobs that perform both nearest neighbor as well as all-to-all personalized collective communication patterns. The details of the workload parameters and distributions can be found in [5]; however, in short, jobs arrive according to a Poisson arrival process, the initial runtimes of the jobs are exponentially distributed, and the widths of the jobs are uniformly distributed on the interval 5 to 20 nodes.

Bandwidth-aware Co-allocation

We have made use of a backfilling meta-scheduler that schedules jobs in three primary phases: local, remote (migration), and co-allocated execution. After the scheduler has backfilled as many jobs as possible via local execution and remote job migration, it begins considering the remaining jobs for co-allocation. The strategy used in this paper ensures that no inter-cluster saturation occurs due to co-allocated jobs in the ideal case, i.e. when the estimates are indeed 100% accurate. (Note that saturation can occur when inaccurate user predictions of PPBW are used.) This strategy is an on-line algorithm, in that each time a scheduling decision is made, the scheduler only has access to the information related to the jobs that have arrived thus far, i.e. not for jobs that will arrive in the future. The interested reader may wish to explore the implementation, performance and limitations of schedulers that relax this constraint in [5].

ADDRESSING ESTIMATE INACCURACIES

User-provided *runtime* estimates are typically inaccurate [18] and have been shown to have an approximately uniform distribution [19] based on several workload traces. Therefore, we make the assumption that user-provided *bandwidth requirement* estimates are also uniformly distributed about their actual value, as there are presently no workload traces that provide these values. For example, a ±80% error in bandwidth predictions means that the estimates are uniformly distributed across the interval $[.2 * PPBW, 1.8 * PPBW]$, where $PPBW$ is the actual per-processor bandwidth of the jobs. This error distribution contains both over and underestimations. We have also included the equivalent root-mean-square percent error (RMSPE) as a secondary x-axis where possible for the reader's convenience. $ERR \sim$ UNIF(-A:A) $\Rightarrow$ RMS(ERR) = $A/\sqrt(3)$ For example, ±80% error is approximately equivalent to 46 RMSPE. RMSPE values are preferable to averages in this case, since the the average of ERR is 0 when uniformly distributed across the interval (-A:A). It is worth noting here that "error" is not an artifact determined *from* the simulation, but rather a modeled input parameter *to* the simulation.

Results and Observations

As mentioned before, determining the effect of inaccurate user predictions on performance ultimately becomes a question of how sensitive the scheduling strategy is to the bandwidth parameter itself. Clearly, co-allocation is affected by the amount of inter-process communication that takes place; therefore, we have setup three experiments to illustrate the effect of different per processor bandwidths (PPBWs). In addition to varying the amount of error in the estimates (and average PPBWs), we also wish to show the difference between making use of estimates that are inaccurate versus not making use of the information in the first place. Therefore we also conducted experiments in which varying percentages of jobs expose their communication requirement to the scheduler. We call this "*information availability*", i.e. the percentage of jobs in the workload stream that reveal their PPBW to the scheduler. When this information is not available, the scheduler simply resorts to local and remote (via migration) job dispatching and does not attempt to co-allocate the given job.

In Figures 1, 3, and 5 we have explored both the error and information availability parameter space for three distinct PPBW intensities, specifically at 150 (low), 300 (medium), and 400 (high) Mbps. Figures 2, 4, and 6 help to complement the 3D plots by focusing on the behavior at four distinct error levels, namely, 0, 50, 80, and 100%. Finally, Figures 8, 7, and 9

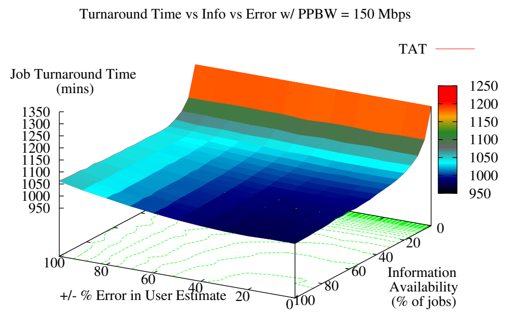

Figure 1. Turnaround time as function of inaccuracy in user-provided bandwidth estimates and information availability at **150** Mbps PPBW. *Note the relative insensitivity to error.*

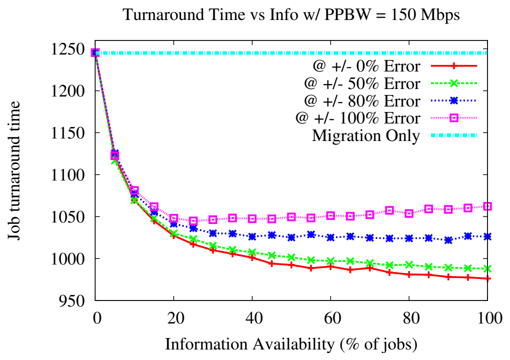

Figure 2. Turnaround time as function of information availability at four distinct levels of inaccuracy in user-provided bandwidth estimates at **150** Mbps PPBW. *Note that at the 100% error level, response is fairly close to the ideal case of 0% error.*

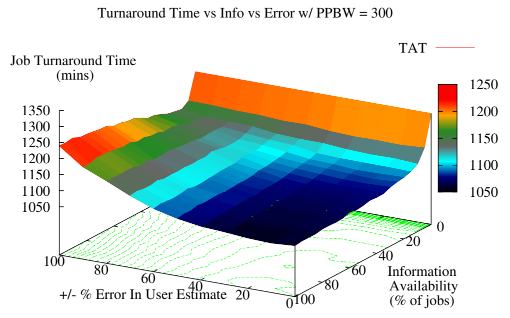

Figure 3. Response at **300** Mbps PPBW. *Note the increased sensitivity to error.*

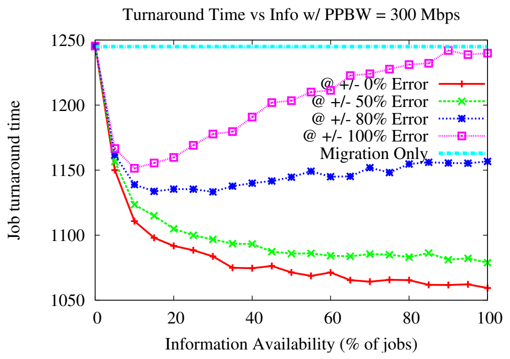

Figure 4. Response at **300** Mbps PPBW. *Note non-monotonic behavior as more jobs expose inaccurate estimates.*

summarize the performance across the error dimension. Note that in each of the 2D plots, the horizontal line "Migration Only" denotes the performance when co-allocation is disabled.

Initial Impact of Inaccuracy

From these results, we can make several interesting observations. The level of "acceptable" error in user estimates is highly dependent on the intensity of inter-process communication. When the parallel job workload exhibits on average per-processor bandwidth of 150 Mbps

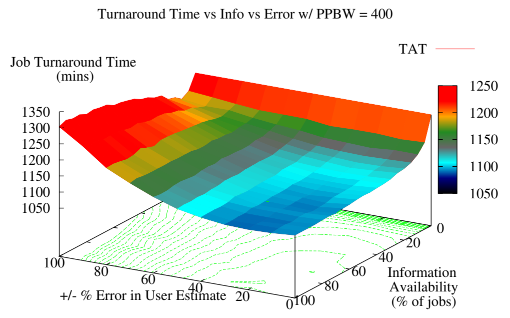

Figure 5. Response at **400** Mbps PPBW. *Yet higher sensitivity to error.*

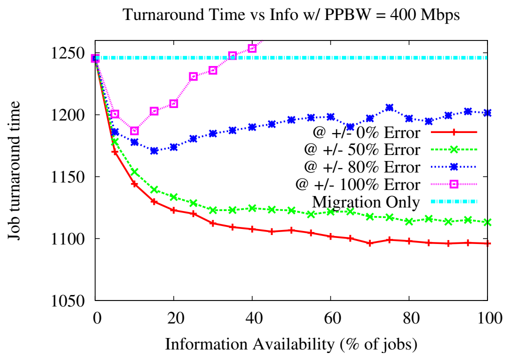

Figure 6. Response at **400** Mbps PPBW. *Yet higher sensitivity to error.*

(Figures 1, 2), even a ±100% error (57.7 RMSPE) in user-predicted bandwidth results in a significant improvement in job turnaround time beyond that of Migration Only. For example, when at least 20% of the workload arrives with user-estimates provided to the scheduler, a 16% improvement over Migration Only is obtained. This is not much less than the 22% improvement [‡] obtained in the ideal case, where 100% of the jobs arrive with perfectly accurate PPBW estimates. In fact, even at error levels up to 100%, the average job turnaround times are essentially monotonically non-increasing functions of information availability. This would correspond to the notion that more information is "always" better, even in the presence of large errors in user predictions of per-processor bandwidth.

However, as the PPBW increases to 300 Mbps (Figures 3, 4), the impact of error in user estimates becomes more severe. In this case, an error larger than roughly 75% actually begins to degrade performance as more information is made available (note the "@ 80% Error" curve in Figure 4). Note that the increase in PPBW results in a max improvement over Migration Only of 16% (in the ideal case) versus the 22% in the 150 Mbps case. Therefore, at the 80% error level, the improvement over "Migration Only" is 8%, representing only 50% of the max improvement possible. This is in contrast to the 73%[2] of max improvement obtained when the PPBW is 150 Mbps.

At the higher PPBW intensity of 400 Mbps (Figures 5, 6), the same error level of 80% referenced above results in only a 4% improvement over Migration Only. This represents roughly 33% of the max improvement obtainable in the ideal case (12% in this instance). However, modest error in user estimates (below 50%) results in improvement that is nearly as good as the ideal case.

[‡]Note that since 16 is actually 73% of 22, the 16% quoted above represents an improvement that is 73% of the maximum improvement obtained in the ideal case. It is important to consider both measures, since the increase in PPBW causes the total possible improvement in the ideal case to decrease.

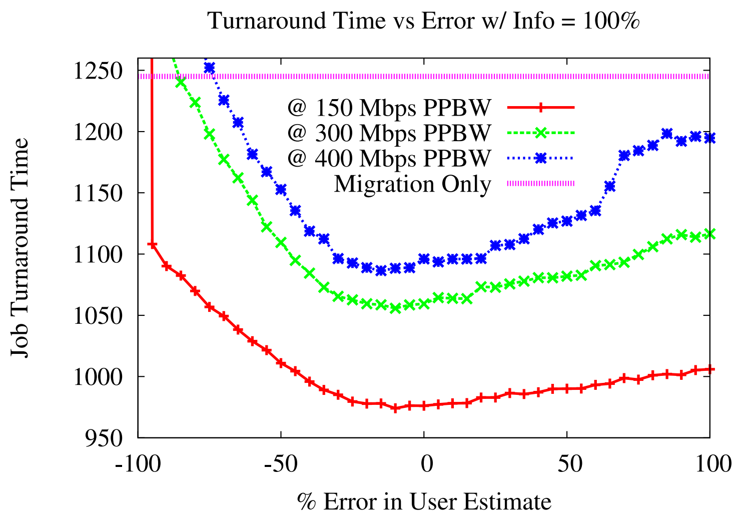

Figure 7. Overestimation vs. underestimation. The error is swept from -100% to 100%. In this experiment, for each simulation run at a given % error, *every* job arrives with the same % error in its bandwidth estimate. *Note that underestimation (error $<$ 0%) is worse than overestimation (error $>$ 0%) with respect to job turnaround time.*

Overestimation versus Underestimation

In the previous experiments, the error in user estimates was uniformly distributed around their actual value; therefore, they represent mixtures of both over and underestimations. One question that quickly arises is whether underestimation is significantly worse than overestimation. In order to address this question, we ran another set of experiments where the error in user-predicted PPBWs was swept from -100% to 100% i.e. from 0 to 2 times their actual value. Figure 7 illustrates the results obtained at PPBWs of 150, 300, and 400 Mbps.

Underestimating the PPBW is significantly worse than overestimation due to the fact that network saturation occurs in this case (Figure 7). However, when an underestimation occurs that results in network saturation, the scheduler becomes aware of this during its next scheduling iteration. It then discounts any clusters with saturated network links as candidates for co-allocation and thereby avoids exacerbating the problem for the duration of the congestion. To some extent, this helps mitigate the impact of underestimation.

On the other hand, the impact of overestimation is not too severe. In this case, the degradation in performance is primarily due to capacity loss, i.e. the scheduler underutilizes network resources that are available for co-allocation. Since co-allocated jobs never experience network congestion during underestimation, job slowdown never occurs.

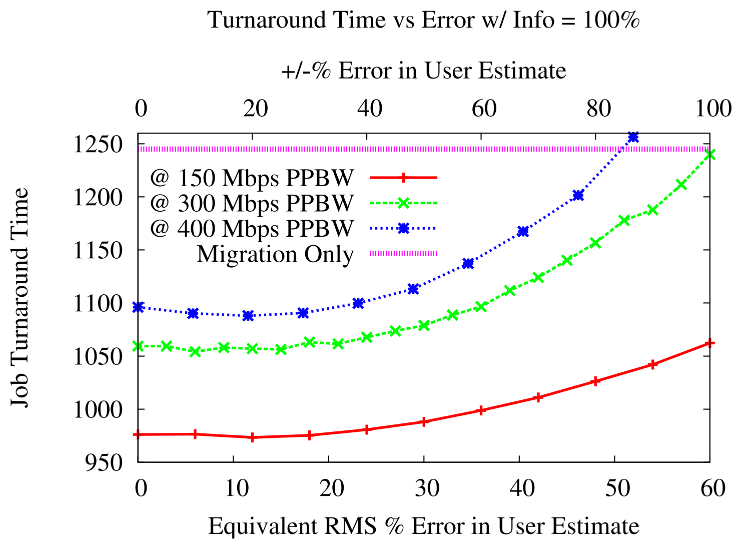

Figure 8. This experiment summarizes the *aggregate* impact of error in that the error modeled in the bandwidth estimates is uniformly distributed both above and below its actual value. As such, it includes both overestimations as well as underestimations; *a more realistic view of overall impact.*

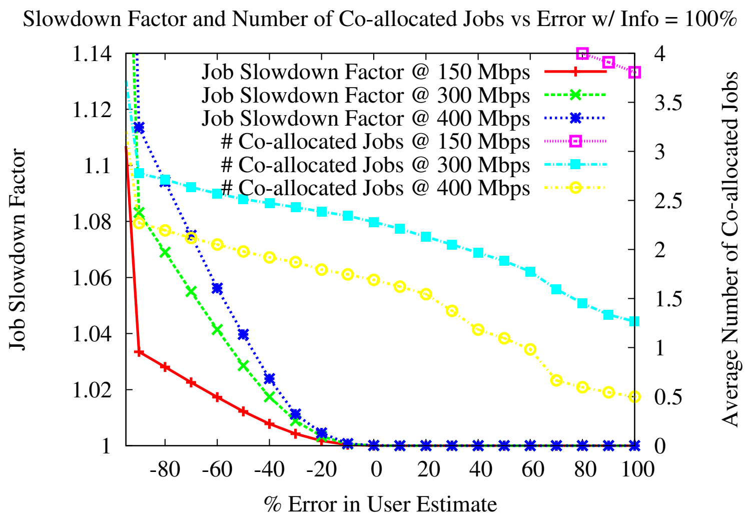

Figure 9. Intermediate causes of degraded performance. *The job slowdown factor is 0 for error* $> 0\%$ *i.e. during overestimation, as seen in the lower three curves.*

Note that at -100% error, the user-specified PPBW is 0. This leads the scheduler to the conclusion that job co-allocation would not require any inter-cluster network bandwidth. This effect can be seen in Figure 7 at -100% error.

Intermediate Causes of Degraded Performance

In the previous experiments, we have demonstrated that inaccuracy in bandwidth estimates leads to various levels of degraded performance, depending on the intensity of inter-process communication and the degree of error. However, what chain of events leads to this degradation?

When the scheduler co-allocates a job that results in network saturation, its runtime (along with any other simultaneously co-allocated jobs that share common links) is lengthened due to the bottlenecks created in the interconnection network. Figure 9 illustrates that the extent of this slowdown reaches as high as 1.12 (12%) when the error level is at -95% (underestimation). This may not appear significant; however note that this 12% is an average across all co-allocated jobs. The integrated time-average of the number of simultaneously co-allocated jobs is provided as an overlay in Figure 9. As the degree of overestimation increases (error going from 0% to 100%), the average number of co-allocated jobs drops from the ideal (at 0% error); consequently leading to capacity loss. Conversely, as the degree of underestimation increases (error going from 0% to -100%), the number of co-allocated jobs increases, as does the job slowdown factor. This indicates lengthened execution times for co-allocated jobs, which in turn results in longer job turnaround times.

ADDRESSING CHECKPOINT/RESTART OVERHEAD

Checkpointing a job requires time to save the state of the application to disk. The amount of time checkpointing requires depends on a large number of interacting factors, but is primarily bound by the amount of memory that needs to be saved and the speed at which it can be written to disk. Later, when the job is to be restarted, time is spent restoring the application image from secondary storage to memory. Altogether, this additional time constitutes an overhead associated with performing checkpointing that would not otherwise be present. In order to provide additional realism, we take this overhead into account when modeling the execution times of checkpointed jobs. In particular, we make this checkpoint/restart time a configurable parameter in order to study the benefit of checkpoint-enabled multi-site scheduling as a function of increasing overhead. In this way, we can determine the extent to which checkpointing is a meaningful tool in recovering from poor scheduling decisions.

In our simulations, we have used checkpoint/restart overhead ranging from 0 minutes (effectively no overhead) to as much as 90 minutes. Most systems with a large amount of RAM and relatively slow disk I/O should be able to write out the entire contents of memory (in the worst case) to disk and read it back in under 90 minutes [20]. For example, 12 minutes is used as an upper bound in [15].

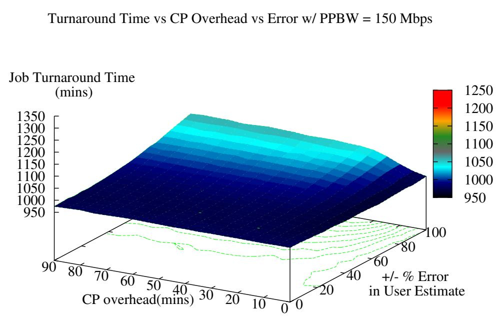

Figure 10. Turnaround time as a function of inaccuracy and checkpoint/restart overhead at **150** Mbps PPBW. *Note the relative insensitivity to error and checkpoint overhead.*

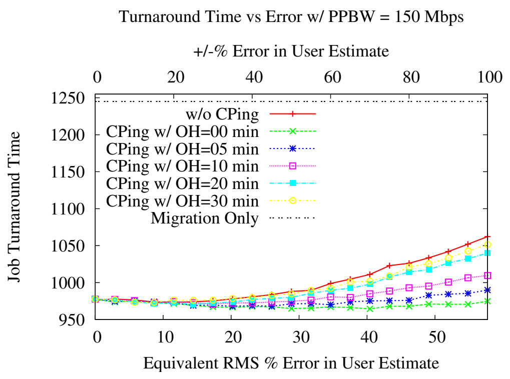

Figure 11. Turnaround time as a function of inaccuracy at five distinct levels of checkpoint/restart overhead at **150** Mbps PPBW.

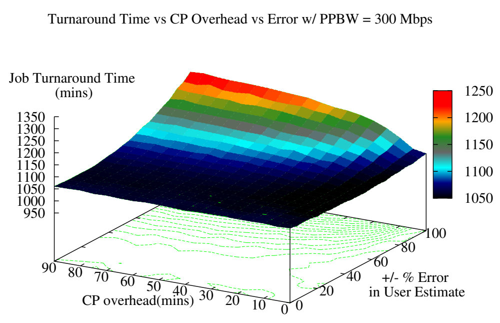

Figure 12. Response at **300** Mbps PPBW.

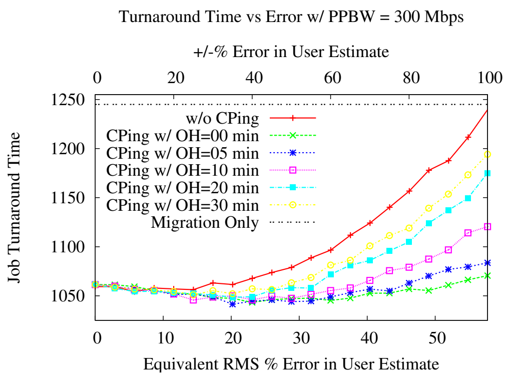

Figure 13. Response at **300** Mbps PPBW.

Results and Observations

In Figures 10, 12, and 14 we have explored both the error and checkpoint overhead parameter space for three distinct PPBW intensities, specifically at 150 (low), 300 (medium), and 400 (high) Mbps. Figures 11, 13, and 15 help to complement the 3D plots by focusing on the behavior at five distinct checkpoint overhead levels, namely, 0, 5, 10, 20, and 30 minutes. For example, a checkpoint overhead of 30 minutes indicates the time to both save the state of the job during the initial checkpoint, as well as the time to restore the state later. For simplicity, we assume the overhead is divided equally between the save and restore. Finally, Figures 17, 18, and 19 summarize the performance across the checkpoint overhead dimension for two error levels. In these plots, we illustrate the difference between the performance at these error levels

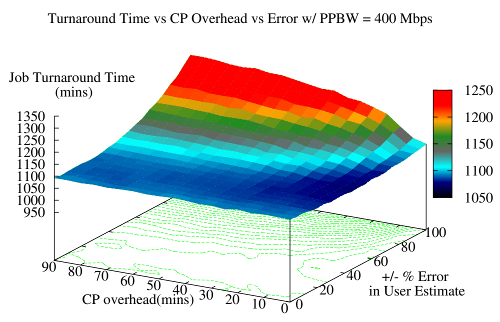

Figure 14. Response at **400** Mbps PPBW.

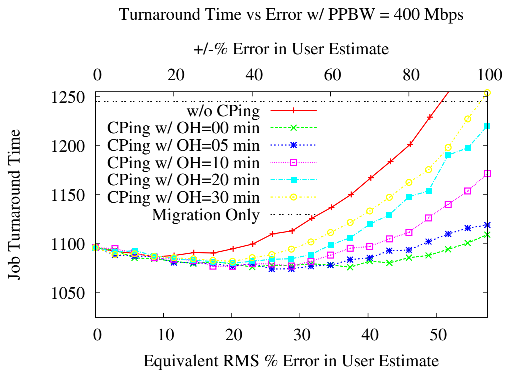

Figure 15. Response at **400** Mbps PPBW. *Note that at moderate levels of overhead (around 5-10 minutes) the process of checkpointing, termination, migration, and restart provides a dramatic improvement.*

with checkpointing enabled as well as disabled for comparative purposes. Note that in each of the 2D plots, the horizontal line "Migration Only" denotes the performance when job migration is allowed, but with co-allocation is disabled.

Impact of Inaccuracies After Enabling Checkpointing

As previously mentioned, error in user estimates may have a significant impact on performance, particularly for jobs that exhibit high intensity inter-process communication. However, when checkpointing, termination, migration, restart (CTMR) is enabled, the negative impact of estimate inaccuracy is greatly mitigated. For example, in the 150 Mbps case (at 100% error), CTMR with a modest overhead of 10 minutes improves the performance of co-allocation from 16% (without CPing) to 19% (with CPing) out of a maximum of 22% improvement (in the 0% error case). As the PPBW increases to 300 Mbps, CTMR with an overhead of 10 minutes improves the performance from 0.8% (without CPing) to 10.5% (with CPing) out of a maximum of 16% (in the 0% error case). Finally, at 400 Mbps, CTMR with an overhead of 10 minutes improves the performance from -4% (without CPing) to +6% (with CPing) out of a maximum of 13% (in the 0% error case). The results for the 5 and 10 minute overhead cases are summarized in Figure 16.

/

What happens to performance as the overhead is increased beyond 10 minutes? From Figures 11, 13, and 15 we see that as the overhead increases, the ability for CTMR to mitigate the effects of estimate error diminishes; an intuitive result. However, at what point does using CTMR become worse than simply ignoring the network over-subscription? Figures 17, 18, and 18 address this question.

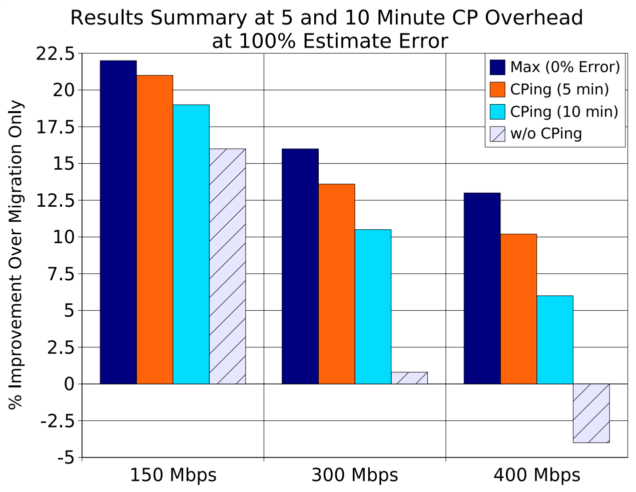

Figure 16. In the 300 and 400 Mbps PPBW cases, enabling CTMR improves the performance by roughly 10% and 14% over the non-checkpointing version at 10 and 5 minute overheads, respectively.

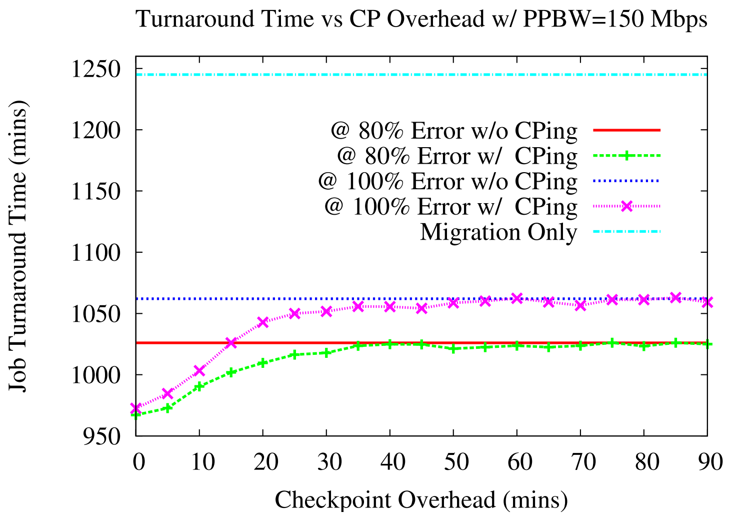

Figure 17. Turnaround time as a function of CTMR overhead at two error levels at **150** Mbps PPBW. *Note that with CTMR enabled, as the overhead increases, the performance asymptotically approaches the performance without checkpointing.*

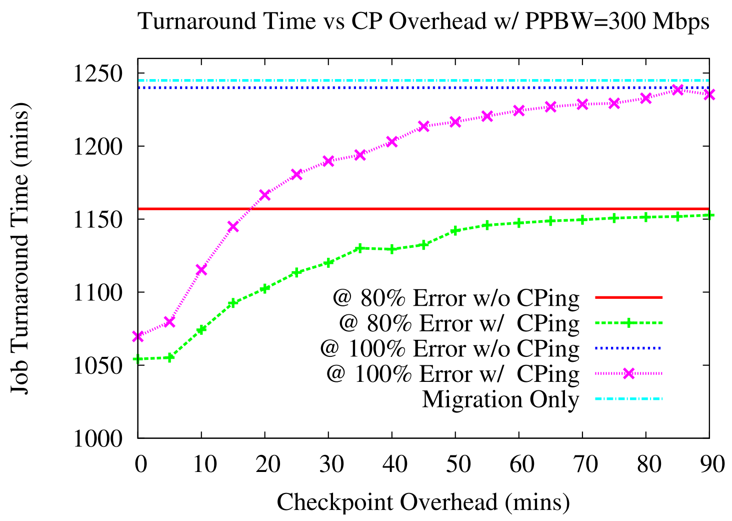

Figure 18. Response at **300** Mbps PPBW. *Note the point of diminishing returns around 20 - 25 minutes of overhead.*

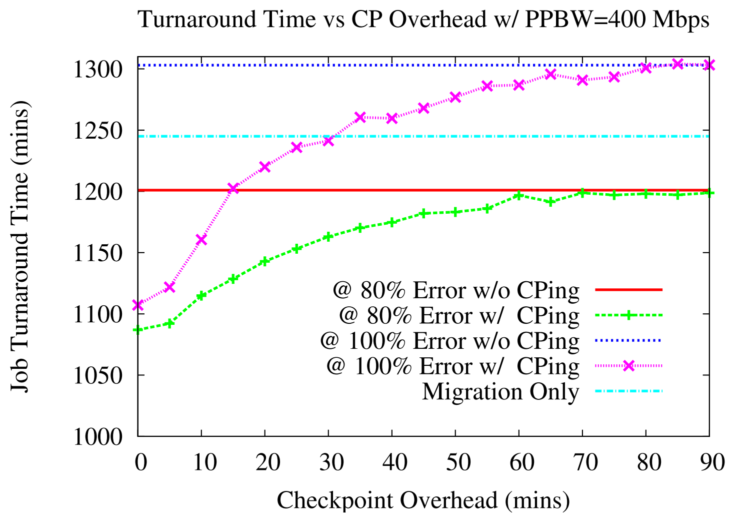

Figure 19. Response at **400** Mbps PPBW. *Note that at 100% error, the performance degrades beyond that of migration only at around 30 minutes of overhead.*

Note that in each figure, the performance with CTMR is never worse than without CTMR for a range of 0 to a generous 90 minutes of overhead. The point of diminishing returns is at roughly 20 - 25 minutes of overhead at each bandwidth intensity. It should be noted that the performance with CTMR does not degrade beyond the performance of Migration Only in the 150 and 300 Mbps cases; however, in the 400 Mbps case (at 100% error), the job turnaround time passes Migration Only at 30.6 minutes of overhead.

CONCLUSIONS

In this paper, we have explored the impact of user-provided bandwidth estimate inaccuracies on overall system performance. We demonstrated that multi-site job scheduling techniques benefit from these estimates, even in the presence of considerable inaccuracy. Furthermore, we have shown that the extent to which these errors impact performance is highly correlated to the inter-process communication intensity. Additionally, we have illustrated that underestimation is more costly than overestimation with respect to average job turnaround time.

Furthermore, we have examined the use of checkpointing followed by job migration in an effort to mitigate the negative impact of user-provided estimate inaccuracy during periods of intense inter-process communication. We have developed a relatively simple checkpoint, termination, migration, restart (CTMR) agent that autonomously decides when to selectively checkpoint jobs to reduce network over-subscription. We subsequently characterize the conditions and the extent to which CTMR improves multi-site parallel job scheduling performance. We demonstrate that CTMR improves performance even when the overhead of doing so is very costly. We show that at moderate levels of overhead, CTMR can be used to greatly mitigate the impact of estimate inaccuracies.

As multi-cluster job scheduling matures, the search for increasingly accurate user estimates will undoubtedly turn to alternative methods of improvement [19, 4]. Users of parallel machines often repeatedly execute the same (or similar) programs; [21]; therefore, future efforts may focus on making use of historical data to predict bandwidth requirements.

ACKNOWLEDGEMENTS

This work was initially supported in part by the Naval Academy Research Council and ORN grant NOOOI407WRZOI02. I would like to thank the initial MGC2007 reviewers as well as the reviewers of this special issue for their collective insight and feedback. Special thanks go to Dr. Walt Ligon and Mr. Brad Settlemyer for use of their Beowulf cluster-based computational mini-grid at Clemson University. Additional thanks go to Mr. Brad Settlemyer at Clemson University as well as Dr. Nathan DeBardeleben at Los Alamos National Laboratory for their editorial feedback.

REFERENCES

1. Jones WM. Using checkpointing to recover from poor multi-site parallel job scheduling decisions. *The 5th Workshop on Middleware for Grid Computing at the ACM/IFIP/USENIX 8th International Middleware Conference*, 2007.
2. Bucar AID, Epema DHJ. The performance of processor co-allocation in multicluster systems. *3rd International Symposium on Cluster Computing and the Grid*, 2003; 302–309.
3. Ernemann C, Hamscher V, Streit A, Yahyapour R. Enhanced algorithms for multi-site scheduling. *Grid Computing - GRID 2002, Third International Workshop, Baltimore, MD, USA, November 18, 2002, Proceedings*, 2002; 219–231.
4. Qin J, Bauer MA. An improved job co-allocation strategy in multiple HPC clusters. *21st International Symposium on High Performance Computing Systems and Applications (HPCS 2007)*, 2007.
5. Jones WM, Pang LW, Stanzione D, Ligon III WB. Characterization of bandwidth-aware meta-schedulers for co-allocating jobs across multiple clusters. *Journal of Supercomputing, Special Issue on the Evaluation of Grid and Cluster Computing Systems*, vol. 34, Springer Science and Business Media B.V, 2005; 135–163.
6. Ngubiri J, van Vliet M, Nijmegen RU. Group-wise performance evaluation of processor co-allocation in multi-cluster systems. *Job Scheduling Strategies for Parallel Processing.* Springer Verlag, 2007. To appear in Lect. Notes Comput. Sci.
7. Weizhe Z, Binxing F, Mingzeng H, Xinran L, Hongli Z, Lei G. Multisite co-allocation scheduling algorithms for parallel jobs in computing grid environments. *Science in China Series F: Information Sciences* 2006; **49**(6):906–926, doi:10.1007/s11432-006-2034-2.
8. Lee CB, Schwartzman Y, Hardy J, Snavely A. Are user runtime estimates inherently inaccurate? *Job Scheduling Strategies for Parallel Processing.* Springer Verlag, 2004; 253–263. Lect. Notes Comput. Sci. vol. 3277.
9. Chiang SH, Arpaci-Dusseau A, Vernon MK. The impact of more accurate requested runtimes on production job scheduling performance. *Job Scheduling Strategies for Parallel Processing.* Springer Verlag, 2002; 103–127. Lect. Notes Comput. Sci. vol. 2537.
10. Jackson D, Snell Q, Clement M. Core algorithms of the Maui scheduler. *Job Scheduling Strategies for Parallel Processing.* Springer Verlag, 2001; 87–102. Lect. Notes Comput. Sci. vol. 2221.
11. Jones WM. The impact of error in user-provided bandwidth estimates on multi-site parallel job scheduling performance. *The 19th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2007)*, 2007.
12. Koo R, Toueg S. Checkpointing and rollback-recovery for distributed systems. *IEEE Transactions on Software Engineering* 1987; **13**:23–31.
13. Deconinck G, Vounckx J, Lauwereins R, Peperstraete JA. Survey of backward error recovery techniques for multicomputers based on checkpointing and rollback. *International Journal of Modeling and Simulation* 1998; .
14. BeoSim Website. http://www.parl.clemson.edu/beosim.
15. Oliner AJ, Sahoo RK, Moreira JE, Gupta M. Performance implications of periodic checkpointing on large-scale cluster systems. *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 18*, IEEE Computer Society: Washington, DC, USA, 2005; 299.2, doi:http://dx.doi.org/10.1109/IPDPS.2005.337.
16. Jones WM, Daly JT, DeBardeleben NA. Application resilience: Making progress in spite of failure. *The Workshop on Resiliency in High-Performance Computing held in conjunction with the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid2008)*, 2008.
17. Daly JT. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Computer Systems* Janurary 2006; **22**:303–312.
18. Weil AM, Feitelson DG. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transactions Parallel Distributed Systems* 2001; **12**(6):529–543.
19. Tsafrir D, Etsion Y, Feitelson DG. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transactions on Parallel and Distributed Systems* 6 2007; **18**:789–803.
20. Plank JS, Thomason MG. Processor allocation and checkpoint interval selection in cluster computing systems. *J. Parallel Distrib. Comput.* 2001; **61**(11):1570–1590, doi:http://dx.doi.org/10.1006/jpdc.2001.1757.
21. Feitelson DG, Nitzberg B. Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. *Job Scheduling Strategies for Parallel Processing.* Springer-Verlag, 1995; 337–360. Lect. Notes Comput. Sci. vol. 949.