Using Checkpointing to Recover from Poor Multi-site Parallel Job Scheduling Decisions

William M. Jones
Electrical and Computer Engineering Department
United States Naval Academy
Mail Stop 14B, 105 Maryland Avenue, Annapolis, MD, USA, 21402-5025

ABSTRACT

Recent research in multi-site parallel job scheduling leverages user-provided estimates of job communication characteristics to effectively partition the job across multiple clusters. Previous research addressed the impact of inaccuracies in these estimates on overall system performance and found that multi-site scheduling techniques benefit from these estimates, even in the presence of considerable inaccuracy. While these results are encouraging, there are many instances where these errors result in poor scheduling decisions that cause network over-subscription. This situation can lead to significantly degraded application runtime performance and turnaround time.

In this paper, we explore the use of job checkpointing to selectively stop offending jobs in order 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 checkpointing improves overall performance. We demonstrate that checkpointing is beneficial even when the overhead of doing so is costly.

Categories and Subject Descriptors

C.2 [**Computer Communication Networks**]: Distributed Systems; C.4 [**Performance of Systems**]: Performance Attributes

General Terms

Job scheduling, checkpointing, clusters, computational grid

1. 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 [1]. This configuration allows considerable flexibility for distributing parallel jobs among available clusters; however, it also dramatically increases the complexity of effectively 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 scheduling [2], [8]. 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 [4, 5, 11] i [1].

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. Previous research in this area has shown that the impact of these inaccuracies range from negligible to severe, depending on a number of factors including the relative intensity of inter-process communication [3]. It is therefore important to identify mechanisms to mitigate the negative impact of these inaccuracies when they occur.

While checkpointing is largely used in parallel and distributed computing to recover from *component failures*, we use it 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 slowdown. In this paper, we therefore focus on job checkpointing alleviate network over-subscription and therefore mitigate job slowdown.

We describe a checkpointing agent that autonomously decides when to checkpoint jobs and provide a rationale behind its parametrization. We subsequently characterize the conditions and the extent to which checkpointing improves multi-site parallel job scheduling performance. We demonstrate that checkpointing improves performance even when the overhead of doing so is very costly. We show that at moderate levels of overhead, checkpointing can be used to mitigate the negative impact of estimate inaccuracies.

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

2. 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 [4].

2.1 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 nodes. Each cluster has its own internal switch. Additionally, the clusters are connected to one another through 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 in order 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.

2.2 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 in order 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.

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.

3. CHECKPOINTING

Checkpointing is largely used in parallel and distributed computing as a mechanism to recover from failures in system components. As the size of parallel systems increase, the mean time between failures (MTBF) typically decreases. 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.

We are primarily interested in checkpointing as a means by which to recover from poor scheduling decisions in order to alleviate network congestion, and therefore mitigate job slowdown. In this section, we describe the checkpointing agent and the motivation behind certain decisions regarding its implementation.

3.1 Checkpointing Agent Motivation

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

Firstly, previous research has shown that occasional over-subscription can actually improve overall job throughput (refer to [4], Section 6.3). For example, the reduction in queue waiting time can outweigh the increase in execution time caused by network saturation. Secondly, 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 there are other scheduling constraints that prevent their 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, one must consider the additional time required to checkpoint/recover. 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.

3.2 Agent Implementation

Each of the above factors, among others, plays an important role in the implementation of our checkpointing 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 checkpoint 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 it's bandwidth requirements, **(2)** the time required to CP (if known) the job is less than some fraction if 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 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 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 it's 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 is not known or unavailable, we assume a generous overhead of 30 minutes.

4. SIMULATION

In this section we describe the simulation parameters and multi-cluster setup. Each of the four clusters in the grid consists 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 [4]; however, in short, jobs are arriving 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.

4.1 Addressing Estimate Inaccuracies

User-provided *runtime* estimates are typically inaccurate [10] and have been shown to have a rather uniform distribution [9] 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).

4.2 Addressing Checkpoint Overhead

Checkpointing a job requires time to save the state of the application to disk. The amount of time this 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 may 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/recover 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 [7]. For example, 12 minutes is used as an upper bound in [6].

4.3 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 however 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 [4].

5. RESULTS AND OBSERVATIONS

In Figures 1, 3, and 5 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 2, 4, and 6 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 8, 9, and 10 summarize the performance across the checkpoint overhead dimension for two error levels. In these plots, we illustrate the difference between the performance

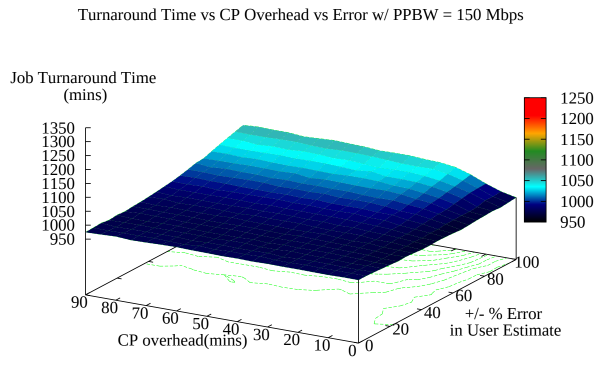

Figure 1: Turnaround time as a function of inaccuracy and checkpointing overhead at 150 Mbps PPBW. ***Note the relative insensitivity to error and checkpoint overhead.***

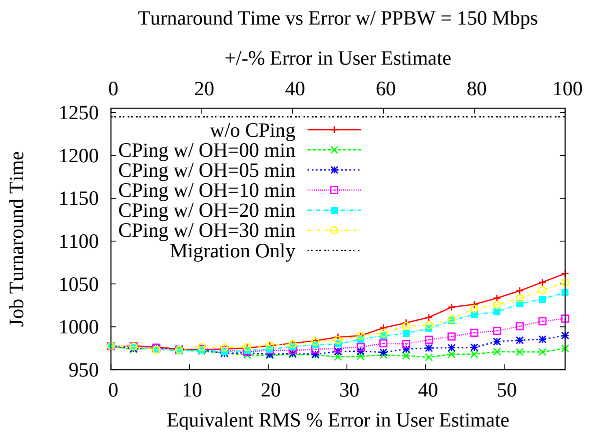

Figure 2: Turnaround time as a function of inaccuracy at five distinct levels of checkpoint overhead at 150 Mbps PPBW.

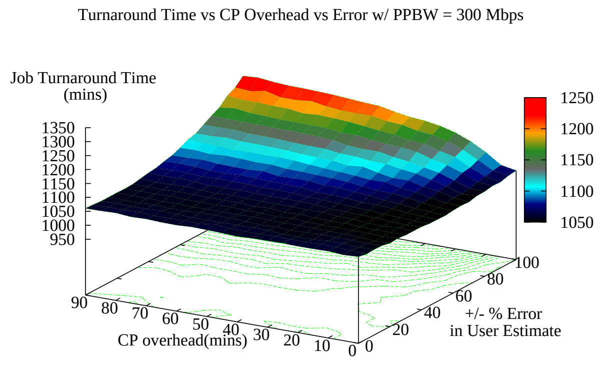

Figure 3: Response at 300 Mbps PPBW.

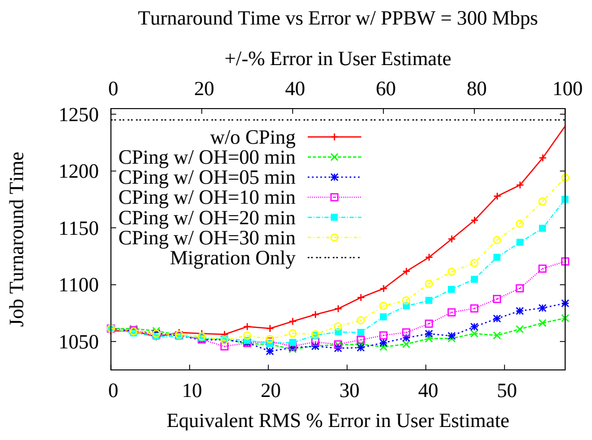

Figure 4: Response at 300 Mbps PPBW.

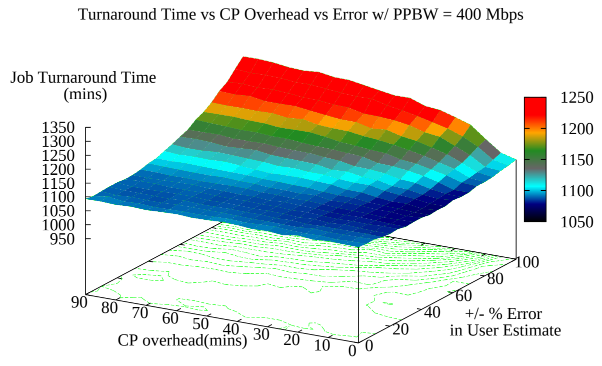

Figure 5: Response at 400 Mbps PPBW.

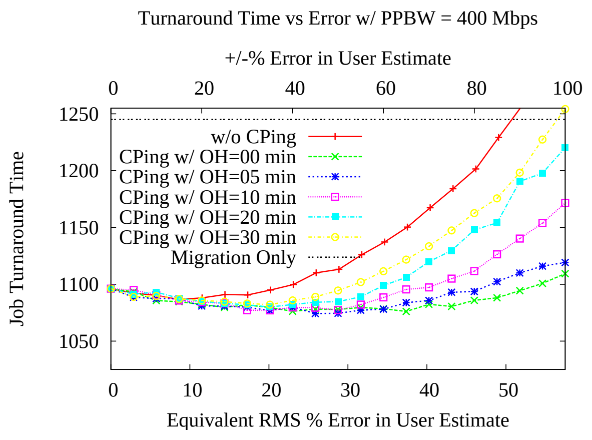

Figure 6: Response at 400 Mbps PPBW. ***Note that at moderate levels of overhead (around 5-10 minutes) checkpointing provides a dramatic improvement.***

at these error levels 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.

5.1 Impact of Inaccuracy

From these results, we can make several interesting observations. As previously noted in [3], 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 (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. Even without checkpointing enabled, a 16% improvement over Migration Only is obtained. This is not much less than the 22% improvement [2] obtained in the ideal case, where 100% of the jobs arrive with perfectly accurate PPBW estimates. As the PPBW increases to 300 Mbps (Figures 3, 4), the impact of error in user estimates becomes more severe. 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. It is also worth noting that there is little benefit to performing co-allocation at the 100% error level (note upper right-hand corner of Figure 4). At the higher PPBW intensity of 400 Mbps (Figures 3, 4), the performance of co-allocation degrades beyond that of Migration Only when the error passes roughly 90%. At 100% error, co-allocation actually results in a 4% *degradation* in performance over Migration Only.

5.2 Impact After Enabling Checkpointing

However, when checkpointing (CPing) is enabled, the negative impact of estimate inaccuracy is greatly mitigated. For example, in the 150 Mbps case (at 100% error), checkpointing 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, checkpointing 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, checkpointing 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 7.

What happens to performance as the overhead is increased beyond 10 minutes? From Figures 2, 4, and 6 we see that as the overhead increases, the ability for checkpointing to mitigate the effects of estimate error diminishes; an intuitive result. However, at what point does using checkpointing become worse than simply ignoring the network over-subscription? Figures 8, 9, and 9 help answer this question.

Note that in each figure, the performance with checkpointing is never worse than without checkpointing for a range of

[2]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.

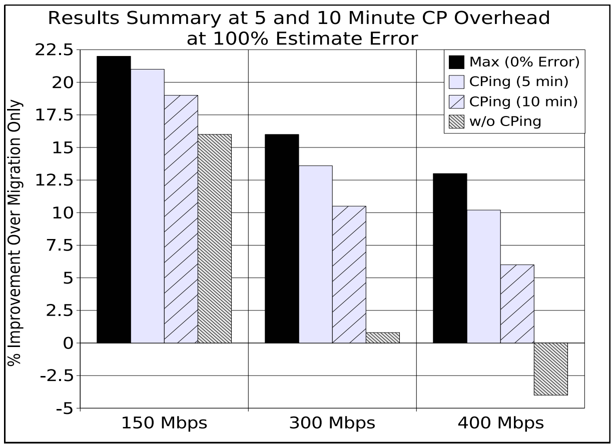

Figure 7: In the 300 and 400 Mbps PPBW cases, enabling checkpointing improves the performance by roughly 10% and 14% over the non-checkpointing version at 10 and 5 minute overheads, respectively.

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 checkpointing 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 around 30 minutes of overhead.

6. CONCLUSIONS AND FUTURE WORK

We have examined the use of checkpointing to mitigate the negative impact of user-provided estimate inaccuracy. We have developed a relatively simple checkpoint agent that autonomously decides when to selectively checkpoint jobs to reduce network over-subscription. We subsequently characterize the conditions and the extent to which checkpointing improves multi-site parallel job scheduling performance. We demonstrate that checkpointing improves performance even when the overhead of doing so is very costly. We show that at moderate levels of overhead, checkpointing 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 [9, 8]. Users of parallel machines often repeatedly execute the same (or similar) programs; therefore, future efforts may focus on making use of historical data to predict bandwidth requirements.

7. ACKNOWLEDGMENTS

This work was supported by the Naval Academy Research Council and ORN grant NOOOI407WRZOI02. 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 Dr. Nathan DeBardeleben at Los Alamos National Laboratory for his editorial feedback.

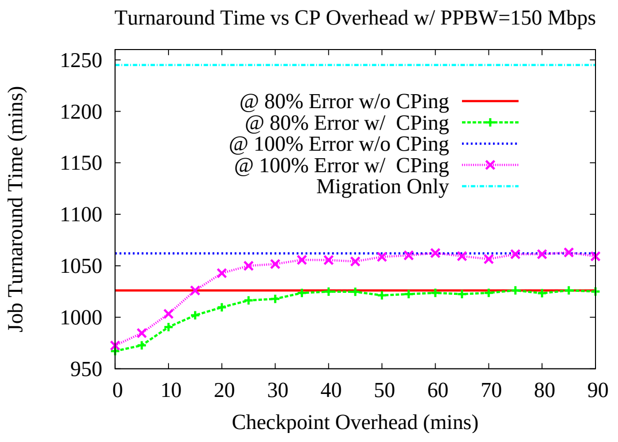

Figure 8: Turnaround time as a function of checkpoint overhead at two error levels at 150 Mbps PPBW. ***Note that with checkpointing enabled, as the overhead increases, the performance asymptotically approaches the performance without checkpointing.***

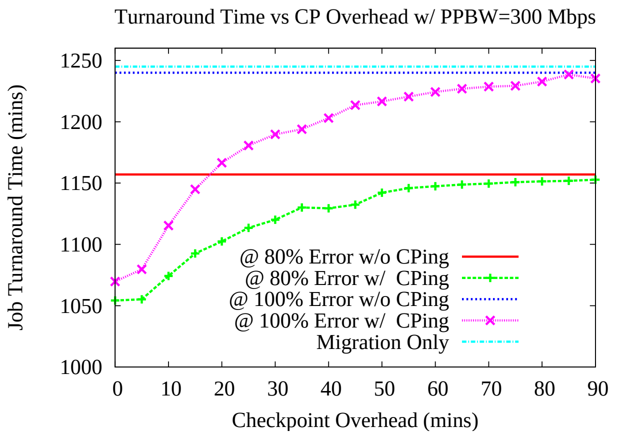

Figure 9: Response at 300 Mbps PPBW. ***Note the point of diminishing returns around 20 - 25 minutes of overhead.***

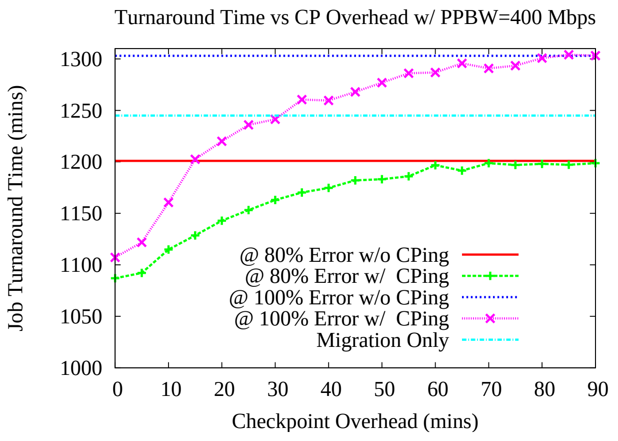

Figure 10: Response at 400 Mbps PPBW. ***Note that at 100% error, the performance degrades beyond that of migration only at around 30 minutes of overhead.***

8. REFERENCES

[1] 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.

[2] 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.

[3] W. M. Jones. The impact of error in user-provided bandwidth estimates on multi-site parallel job scheduling performance. In *The 19th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2007), to appear* November 2007.

[4] W. M. Jones, L. W. Pang, D. Stanzione, and W. B. Ligon III. Characterization of bandwidth-aware meta-schedulers for co-allocating jobs across multiple clusters. In *Journal of Supercomputing, Special Issue on the Evaluation of Grid and Cluster Computing Systems*, volume 34, pages 135–163. Springer Science and Business Media B.V, November 2005.

[5] J. Ngubiri, M. van Vliet, and R. U. Nijmegen. Group-wise performance evaluation of processor co-allocation in multi-cluster systems. In *Job Scheduling Strategies for Parallel Processing*. Springer Verlag, 2007. to appear in Lect. Notes Comput. Sci.

[6] A. J. Oliner, R. K. Sahoo, J. E. Moreira, and M. Gupta. Performance implications of periodic checkpointing on large-scale cluster systems. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 18*, page 299.2, Washington, DC, USA, 2005. IEEE Computer Society.

[7] J. S. Plank and M. G. Thomason. Processor allocation and checkpoint interval selection in cluster computing systems. *J. Parallel Distrib. Comput.*, 61(11):1570–1590, 2001.

[8] J. Qin and M. A. Bauer. An improved job co-allocation strategy in multiple HPC clusters. In *21st International Symposium on High Performance Computing Systems and Applications (HPCS 2007)*, May 2007.

[9] D. Tsafrir, Y. Etsion, and D. G. Feitelson. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transactions on Parallel and Distributed Systems*, 18:789–803, 6 2007.

[10] A. M. Weil and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transactions Parallel Distributed Systems*, 12(6):529–543, 2001.

[11] Z. Weizhe, F. Binxing, H. Mingzeng, L. Xinran, Z. Hongli, and G. Lei. Multisite co-allocation scheduling algorithms for parallel jobs in computing grid environments. *Science in China Series F: Information Sciences*, 49(6):906–926, 2006.