Application Monitoring and Checkpointing in HPC: Looking Towards Exascale Systems

William M. Jones*
Department of Computer Science
Coastal Carolina University†
Conway, SC, USA
wjones@coastal.edu

John T. Daly
Center for Exceptional Computing
Department of Defense, ACS†
Fort Meade, MD, USA
john.t.daly@ugov.gov

Nathan DeBardeleben
High Performance Computing
Los Alamos National Laboratory†
Los Alamos, MN, USA
ndebard@lanl.gov

ABSTRACT

As computational cluster computers rapidly grow in both size and complexity, system reliability and, in particular, application resilience have become increasingly important factors to consider in maintaining efficiency and providing improved compute performance over predecessor systems. One commonly used mechanism for providing application fault tolerance in parallel systems is the use of checkpointing.

We demonstrate the impact of sub-optimal checkpoint intervals on application efficiency via simulation with real workload data. We find that application efficiency is relatively insensitive to error in estimation of an application's mean time to interrupt (AMTTI), *a parameter central to calculating the optimal checkpoint interval*. This result corroborates the trends predicted by previous analytical models. We also find that erring on the side of overestimation may be preferable to underestimation.

We further discuss how application monitoring and resilience frameworks can benefit from this insensitivity to error in AMTTI estimates. Finally, we discuss the importance of application monitoring at exascale and conclude with a discussion of challenges faced in the use of checkpointing at such extreme scales.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Performance Attributes

*Corresponding author.

†Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the Los Alamos National Laboratory, the US Department of Defense, or Coastal Carolina University.

‡This work builds on the material presented in "The Impact of Sub-optimal Checkpoint Intervals on Application Efficiency in Computational Clusters", in the *19th ACM International Symposium on High Performance Distributed Computing*, Chicago, Illinois, June 20-25, 2010, pp. 276-279.

General Terms

Performance, Reliability

1. INTRODUCTION

Cluster computing has been a viable supercomputing alternative for many different application domains for more than a decade. As the price-to-performance ratio of off-the-shelf computing and networking hardware has continued to decrease, making use of larger clusters to help solve computationally expensive problems has become extremely popular and rather common-place. The desire to improve model fidelity by running larger and more detailed scientific simulations is a constant in what has often become a race to build the world's largest and most powerful cluster computers. While measures such as raw compute performance and system capacity are still at the forefront of our desired cluster characteristics, such issues as system reliability and application resilience have started to become increasingly important factors when evaluating overall cluster viability, particularly as we look towards exascale, i.e., systems capable of $O(10^{18})$ floating point operations per second.

As the size of a cluster increases, the system mean time between failures (SMTBF) tends to decrease inversely proportional to the number of components and in some cases, even more rapidly when failures exist that simultaneously impact multiple jobs. Applications can experience an interruption in service due to such failures, and as system sizes grow, application failures will become a much more critical issue in addressing overall system performance. In addition to hardware failures, novel system software stacks coupled with legacy parallel scientific applications deployed on modern cluster platforms push the envelope of reliability.

Much work has been done to improve system reliability by attempting to provide redundancy or other fault-tolerant technologies to keep the application from ever being interrupted. While this is a very interesting research area, it is not the focus of this paper. We accept that applications will inevitably be interrupted and are primarily interested in mitigating the overall impact on application and system performance in the face of these interruptions.

One commonly used technique to recover from application failure is the use of checkpointing. During checkpointing, an application writes its entire state to non-volatile secondary storage so that in the event that it is interrupted, it can resume its work from the last checkpoint rather than from the beginning. For applications that use checkpoint restart

as their primary means of fault tolerance, only a fraction of the application's execution time, or *run time* (t_r), is spent performing actual computational work that represents forward progress towards a solution. That time is referred to as *solve time* (t_s). The difference between the solve time and the execution time consists of the dumptime, i.e., the time spent writing out checkpoint data, restarting after an interrupt, and performing rework to move the calculation from the latest checkpoint back to the point where the interrupt occurred. Application efficiency is defined to be the ratio of solve time to run time, t_s/t_r, and as such, ranges from 0.0 to 1.0. While writing and reading the checkpoint data is a type of overhead that consumes valuable system resources, the savings in rework times due to failure can often outweigh the cost of performing checkpointing in the first place.

One aspect of employing checkpointing is properly assigning a checkpoint interval, i.e., the time from the beginning of one checkpoint to the beginning of the next. Given a set of jobs and failure parameters, it is possible to assign this interval in such a way that it maximizes application efficiency [2, 13, 24]. A principal question here is the extent to which sub-optimal interval assignment impacts this efficiency. *This is of particular importance due to the fact that the underlying parameters necessary to optimize the interval width may be difficult to determine or potentially contain a large amount of estimation error.*

In this paper we briefly present results from a simulation-based study using real workload data that demonstrate that application efficiency is relatively insensitive to error in optimal checkpoint interval assignment, and by extension error in estimates of application mean time to interrupt (AMTTI), even in the presence of high system failure rates. These simulation-based results match trends predicted by previous analytical models [2, 13, 24].

Finally, we discuss the importance of application monitoring at exascale and highlight MoJo, a potential candidate system being developed for deployment across three US national labs. We conclude with a discussion of challenges faced at exascale, particularly as they apply to application efficiency and checkpointing as a resilience mechanism.

2. SIMULATION

In this section, we describe the simulation-based study conducted to demonstrate the impact of sub-optimal checkpoint interval assignment on overall application efficiency.

Our cluster simulator [14, 10] is parameterized to model the Pink cluster at Los Alamos National Laboratory (LANL). Pink was a 1024 node Myrinet connected cluster at LANL of which 963 nodes were available for user applications. Each node consisted of two Intel 2.4-GHz Xeon processors. Pink was available for use by external LANL collaborators and was decommissioned in 2008. Pink was also used to develop and test the Beowulf Distributed Process Space (BProc) [9] software for system administration.

The simulator's workload generator is tuned to both the sizes and run times of jobs seen on Pink during an 11-month period in 2007 as seen in Figure 1. We assume that jobs arrive according to a Poisson process, and have adjusted the interarrival times to properly load the system, as is typical in discrete-event driven simulations of parallel job scheduling.

We also assume that the time between system failures are also exponentially distributed. In our simulator, as a failure "arrives", any job impacted by the failure is once again

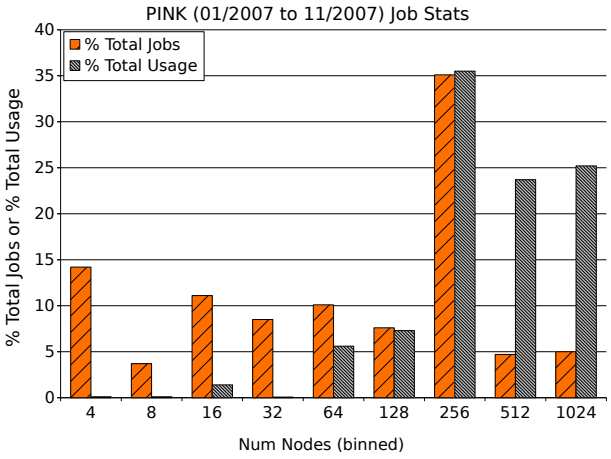

Figure 1: Job size and usage distributions used in the simulation. These are based on a 1-year trace of jobs on LANL's PINK cluster.

placed in the job queue. Once the job is dispatched again, it incurs a restart time as well as the overhead of re-doing the work it had previously done after the last checkpoint.

In [2], Daly demonstrated that the ratio of solve time to execution time is maximized when the checkpoint interval is chosen according to this first order approximation

$$t_c \approx \sqrt{2\delta M} \quad \text{for} \quad \delta < \frac{1}{2}M \ , \tag{1}$$

which agrees with Young's [24] original work in this area. Here δ is the dumptime and M is the AMTTI, which can be estimated based on the SMTBF and the width of the given job [13]. In our simulation, each time a job is dispatched, its checkpoint interval is set according to Equation 1, where δ is set to ten minutes, a reasonable assumption given to total RAM, disk I/O, and network performance of the Pink cluster. Then, by introducing an error in the assignment of the optimal checkpoint interval, we can conduct a parameter sweep as part of our sensitivity analysis to determine the impact to average application efficiency.

3. RESULTS AND OBSERVATIONS

In this section, we present the simulation-based results using BeoSim, a custom event-driven simulator that has been previously used in the study of network aware multi-cluster parallel job scheduling, [11, 13, 12, 14].

By making use of LANL's Pink cluster workload characterization of 50,000 jobs, we we able to conduct a parameter study of average application efficiency, *weighted by the width of the job*, as a function of various failure loads and error in optimal checkpoint interval assignment. The results of this study are summarized in Figure 2 and corroborate the general trends found in the analytical model [2, 13, 24].

As can be seen from the data, at system mean time between failures (SMTBFs) around 30 minutes, an error factor of two results in a drop of only 8% in average application efficiency. *Note that an error factor of two in t_c would actually represent an error factor of four in the estimate of M (AMTTI) due to the quadratic relationship between t_c and M, assuming a relatively fixed value of δ.*

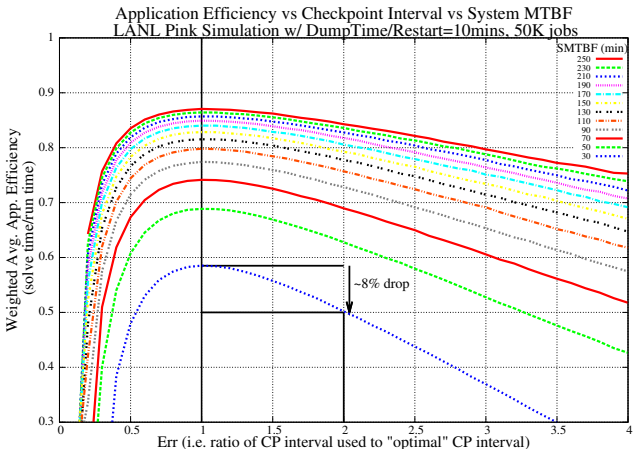

Figure 2: Simulation-based results show the impact of error in optimal checkpoint interval assignment at various system failure rates. *Note that an error factor of two in the interval calculation (i.e., an error factor of four in M (AMTTI) estimation) results in less than a 10% drop in application efficiency.*

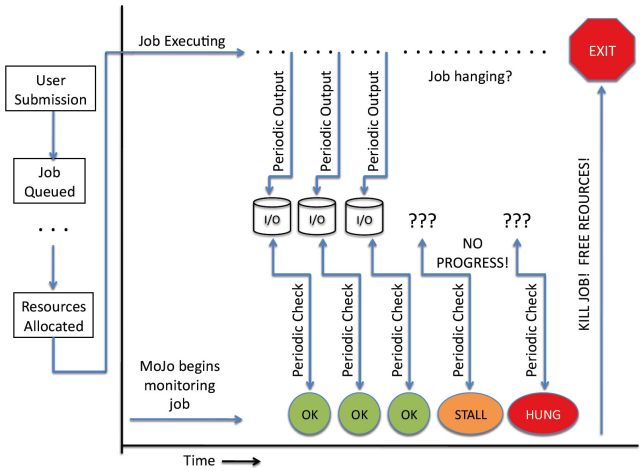

Figure 3: MoJo (MonitorJob) sample operation

This is especially significant considering that a 30-minute SMTBF is extremely severe for a cluster of only 1024 nodes. In a cluster this size, where node failures are probabilistically independent, the individual nodes would need to have a MTBF of slightly over 20 days to result in such a short SMTBF. This suggests that error in optimal interval assignment would likely result in a much smaller impact on application efficiency than what is suggested by the more severe SMTBFs used in the simulation-based study.

From the data we can also see another important trend regarding the choice of application interval. It appears to be far more important to err on the side of overestimation, rather than underestimation. As the checkpoint interval becomes shorter, the impact becomes far more severe.

4. APPLICATION MONITORING

One of the caveats of using the first-order approximation to assign the per-job optimal checkpoint interval is the fact that one must first know the dumptime, i.e., the wall-clock time required to create a checkpoint file, and perhaps more importantly, the AMTTI. Current efforts in application monitoring [18] are beginning to provide *some* of the data necessary to make these estimations. There are many issues to consider here. Firstly, the dumptime depends on the width of the job, the amount of data necessary to save the state, and the speed and utilization of the network, just to name a few. Furthermore, the AMTTI depends not only on the width of the job, but also the resources it is mapped across and their respective failure characteristics. Moreover, hardware tends to fail more often when an application is running on it than when it is idle; therefore, there are dependencies between the applications and the hardware that describe aggregate failure modes.

By having access to a solid application monitoring framework, coupled with system-level monitors, we can begin to extract information over the initial lifetime of a HPC system's usage that will help us more accurately estimate such parameters as dumptime and AMTTI, and in turn, more optimally assign such values as checkpoint intervals. One such application monitoring tool is MoJo (MonitorJob) which is developed as part of the Common Computing Environment program at LANL, Sandia National Laboratory and Lawrence Livermore National Laboratory [1].

MoJo is a new, very unobtrusive approach to application monitoring which works well on the types of production machines and legacy codes that are used as part of the Advanced Simulation and Computing Program of the Department of Energy's National Nuclear Security Administration (NNSA) Defense Program. These supercomputers often cannot be modified with revolutionary kernels or intrusive middleware. Further, the applications run on these architectures are resistant to source code (and even some times binary) instrumentation. This is particularly the case at LANL where some large legacy codes are under extreme scrutiny and cannot be modified in such ways due to organizational directives. MoJo therefore takes a very practical approach at monitoring for application progress by simply watching for application interactions with the file system. This interaction can come in many ways, the simplest of which is merely growth of a data or log file but can also include watching for the creation of new files such as checkpoint or image files. While this approach is fairly naïve, it also has the advantage of being amazingly practical, unintrusive, and extremely portable across a wide range of HPC systems in the NNSA complex.

Jobs offer themselves up for monitoring through a single command executed at application launch which tells MoJo how to measure forward progress for this application and at what periodic rate the application expects to see this progress. Typical workloads will produce some form of output (even in the form of a simple, textual log file) every five to ten minutes. MoJo then monitors the progress and if it detects that the application is not making progress, it can do a variety of things including sending email, paging the user or administrators, killing the application, and even restarting the application. Figure 3 depicts how MoJo monitors an application that begins to stall and then even kills the hung application to free up the resource for another job.

Most important to the discussion of optimal checkpoint intervals is the statistics captured by MoJo when it observes

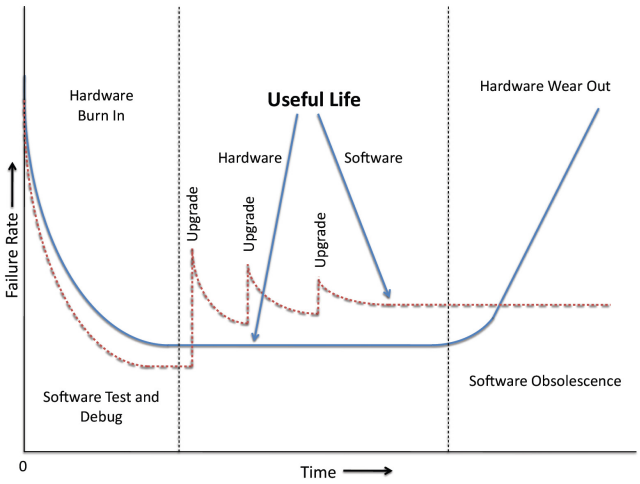

Figure 4: The "bathtub curve" depicting the hardware and software lifetimes of HPC systems. *Note that hardware reliability improves over time and remains fairly constant until end of life when components begin to wear out but software reliability improves piecewise over time but faces periodic downturns because of the bugs and increasing complexity associated with upgrades.*

applications. Information about the status of the application at each periodic interval is recorded in a global database. This database can be mined for such statistics as: how often particular applications run to completion, which sets of hardware are typically associated with stalled or hung jobs, and ultimately, the application mean time to interrupt as seen from the perspective of a tool closely monitoring the application's own definition of forward progress. MoJo provides the framework that gives us the potential to get better data than is currently available in scheduler logs, and if possible, correlate failures at the job level to failures at the node/component level. Some of the extreme scale supercomputers with light-weight kernels do not provide the level of detail at the individual node level necessary to supplement the generic data available in scheduler logs. MoJo attempts to bridge this gap without being as intrusive.

The "bathtub curve", such as Figure 4, is often used to depict hardware component life cycles [19] but it can also be used to depict software. On large HPC systems where hardware vendors test components individually, but rarely as an integrated solution, the burn-in period is seen at deployment rather than in a vendor's laboratory. In particular with the extreme-scale systems, this is due to the fact that it is prohibitively expensive for vendors to assemble and test supercomputers prior to deployment. Most vendors simply do not have facilities large enough to do this type of testing. The problems of integration at scale are also seen by software and are further complicated by software upgrade cycles. While Figure 4 depicts an example of what we think this bathtub curve might look like for a given system, a tool like MoJo can be used to empirically plot this curve based on real failure characteristics seen by applications running on a given system.

Using the simple model proposed in [4] to estimate the AMTTI as a function of the width of the job, one can compute a maximum likelihood estimate, demonstrated in [18], of the AMTTI from data collected with application monitoring. To do so requires gathering several pieces of data from jobs on the system including: the number of nodes allocated to the job, the time that the job spent running and the number of interrupts that occurred during the job run.

The more job data that is collected, the more accurate the estimate of AMTTI becomes. Daly gives examples of these estimates, using simulated data, based on information collected from as few as 100 jobs. For large production systems that run hundreds if not thousands of jobs per day, application monitoring provides more than sufficient data for AMTTI estimates. Furthermore, the data is simple enough to collect, except for the fact that the actual "number of interrupts" may be very difficult to determine from the application's perspective because job status is frequently not returned to the job monitor. If the job monitor only has access to the return status of the resource manager, then it may be difficult or impossible to determine whether a job stopped running normally or because of an interrupt. This is certainly an issue that will need to be addressed in future research on application monitoring.

Another issue that must be addressed in any realtime monitoring of AMTTI is the implicit time dependency of the failure rates. Current estimation techniques, including the one proposed above, assume that the failure rate is constant over the period of observation. So, for instance, if application monitoring collects failure data on 500 jobs that ran over a weekend, the failure rate of the system, and thus the per node AMTTI, is assumed to be constant over the weekend. This is clearly not the case, particularly in cases where system components are performing in some degraded state that causes their failure rate to suddenly increase. Currently, no well-defined calculus exists to address these sorts of issues, though proposals to develop such techniques based on "unknown" stochastic probability functions exist in work like that of Singpurwalla and Wilson [21]. In the case of HPC there is the unique problem that those stochastic probability functions may only be piecewise continuous, with discontinuities corresponding to changes in system state of configuration.

5. LOOKING TOWARDS EXASCALE

Extreme-scale computing is looking next to the challenges of exascale, or roughly 100 times faster than the world's fastest supercomputer (as of November 2011). The DOE, DARPA [15, 22, 7], and other government agencies have been focused on this target and identifying the expected challenges. The International Exascale Software Project (IESP) [6] has been examining the changes needed to operating systems, programming languages, middleware, applications, and other software components of an exascale supercomputer. While all aspects of supercomputing are sure to need serious re-design, we focus primarily on the challenge of building a resilient exascale supercomputer.

The failure rates associated with the strawman architectures presented in the DARPA ExaScale Computing Study [15, 22, 7] predict SMTBF of 35-39 minutes at exascale. Figure 5 makes it clear that global checkpointing of data cannot be the primary means of fault tolerance for capability computing at exascale. The report estimates that dumptimes will be on the order of several hours, meaning the predicted exascale systems in this study will be unable to complete a

checkpoint before seeing component failure.

The DARPA ExaScale Computing Study, however, viewed the storage hierarchy as being similar to what is used in present-day supercomputers - that is, network attached high performance parallel file systems. The DOE's internal exascale planning has taken a different approach. Grider [8] suggests the use of non-volatile memory (NVRAM) at the rack-level where checkpoints are written to by the applications quickly. Then, the checkpoints are asynchronously transferred from NVRAM to the parallel file system at a slower rate. Should failure occur, the checkpoints would be available locally in NVRAM and the application could quickly restart. This approach essentially changes the storage hierarchy by introducing a new type of media. With this approach, it seems possible to write checkpoints in a few minutes, even if the machine as a whole is failing about every 30 minutes as DARPA predicted. This approach assumes that large amounts of rack-level NVRAM will become economical and high performing. Hybrid approaches such as those proposed by Muralimanohar [5] take a similar approach by making use of local non-volatile phase-change RAM to improve checkpointing speed.

Turning to the DARPA UHPC RFI [23] one finds suggestions that in order to meet power and performance requirements associated with exascale, applications will need to achieve up to "billion-way parallelism". Writing a single global checkpoint could take hours. Writing multiple individual checkpoints to local storage, the approach described in DARPA ExaScale Computing Study [15], could be accomplished in seconds to minutes, but until the data is successfully migrated off of the node that data will not be unavailable after a node crash. Managing on the order of a billion checkpoints and using them to reconstruct a consistent application state will be a daunting task.

To further complicate matters, LLNL's experience on Atlas at only 4096 processors [16] demonstrates that application checkpointing to disk can be the single most demanding activity performed by the application. Atlas is an 1152 node Linux cluster at LLNL containing four dual-core AMD Opteron processors and 16 GB of main memory connected by Infiniband 4x links running at DDR. In the case of the **pf3d** applications which ran daily over a seven month time period on 512 nodes, the application team observed a failure every 1.5 hours on average. Although writing a global checkpoint to the Lustre parallel file system took an average of 3.5 minutes, it could take as long as 1.5 hours due to load from other jobs. Thus, the application was configured to checkpoint every few compute cycles and still ran the risk of interruption prior to completion of the next checkpoint.

Moody *et al.* [16, 17] demonstrate that in the case of Atlas, turning off global checkpointing and writing results to RamDisk on neighboring nodes not only decreased the overhead for writing checkpoints to 5 seconds, but caused the SMTBF to increase from 1.5 hours to 1.5 days! This was because the most predominant failure mechanism was network errors. Could such a technique be the silver bullet that resolves the checkpoint restart issue? Certainly it offers many advantages to traditional global checkpointing, but one must not forget that the ratio of solve time to run time for all recovery based approaches to fault-tolerance will still be governed by the equations presented in [2, 24]. Even though checkpointing to a neighboring node can drastically reduce the dumptime, this approach does not address the

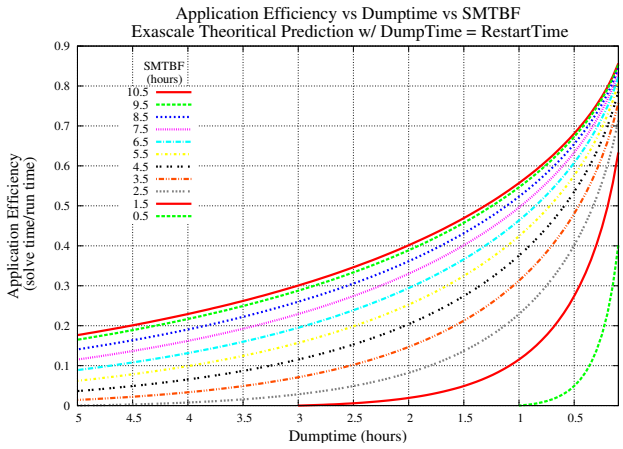

Figure 5: Analytical-based results to demonstrate impact on application efficiency of using checkpointing for one of the DARPA strawman exascale-sized systems. *Note that these exascale supercomputers are capability systems implying that mission critical codes will be run across the entire platform. With over 3PB of RAM, dumptimes will likely be several hours.*

issue of restart time which will be dominated by time to detect and requeue an application as opposed to the time to read the data back out of memory [12].

Also, consider that the strategy of checkpointing to nearest neighbors is only effective if the number of redundant copies of node data is greater than or equal to the number of nodes that go down in a failure. For multi-component failures, which Daly [3] and others [20] have demonstrated contribute significantly to AMTTI, some form of global checkpoint will still be necessary. Thus, this approach requires that some type of storage be over-provisioned on a per node basis in order to provide the necessary redundancy for a recovery. Redundancy costs money and power. Writing to a global file system is more economical in the sense that the disks need to be there anyway, to store application results, but when they are not being used to store results, they can be used to store checkpoints. The point is that systems do not generally need to purchase twice as many disks to accommodate the storing of checkpoints to the filesystem, but if checkpoints are stored in memory [5], then systems will likely need to be purchased with twice as much memory.

Finally, any negative impact on application efficiency due to estimate error in the requisite factors used to determine optimal checkpointing intervals will only be exacerbated by the sheer size and complexity of extreme-scale HPC systems. Although the simulation-based results presented in this paper indicate minimal sensitivity in many realistic situations on the smaller LANL PINK cluster, there is a rapid loss in application efficiency due to increasing dumptimes at extreme scale, particularly at MTBFs predicted by the current DARPA exascale reports (Figure 5). This implies a heightened sensitivity to estimate error, since the MTBF is a significant factor in calculating an optimal checkpoint interval (Equation 1). In the final analysis, there is no "free lunch", and the path towards exascale will likely be a challenging one for any flavor of recovery-based fault tolerance!

6. CONCLUSIONS

We present the results of a simulation-based study that addresses the extent to which error in AMTTI estimates, and by extension the error in checkpoint interval assignments, impacts application efficiency. We show via simulation using workload data that application efficiency is fairly insensitive to these types of errors, even at high system failure rates. These simulation-based results corroborate trends predicted by previous analytical models. Additionally, our results indicate that underestimating an application's optimal checkpoint interval is more severe than overestimation.

Finally, we discuss the importance of application monitoring at exascale and highlight MoJo, a potential candidate system being developed for deployment across three US national labs. We conclude with a discussion of challenges faced at exascale, particularly as they apply to application efficiency and checkpointing as a resilience mechanism.

7. FUTURE WORK

A more sophisticated model of failure processes would help distinguish among failures that directly impact multiple jobs. For example, multiple jobs might be affected by a single switch failure. An analysis of failure types and associated impacts may provide insight into precisely determining the optimal checkpoint interval based on the failure characteristics of the resources it is mapped across [17].

Additionally, addressing "soft failures", i.e., failures that simply degrade performance as opposed to those that catastrophically halt progress, would allow an application monitor to determine whether dynamically reallocating resources based on conditions would in fact lead to improved performance. This is of particular importance in addressing intelligent parallel job scheduling strategies in computational grids where jobs may be migrated [11] and are moldable.

8. REFERENCES

[1] R. Ballance and N. DeBardeleben. The Mojo Application Monitoring Tool Suite. In *11th LCI International Conference on High-Performance Clustered Computing*, March 2010.

[2] J. T. Daly. A Higher Order Estimate of the Optimum Checkpoint Interval for Restart Dumps. *Future Generation Computer Systems*, 22:300–312, 2006.

[3] J. T. Daly. Methodology and metrics for quantifying application throughput. In *Proceedings of the Nuclear Explosives Code Developers Conference*, 2006.

[4] J. T. Daly, L. A. Pritchett-Sheats, and S. E. Michalak. Application MTTFE vs. Platform MTBF: A Fresh Perspective on System Reliability and Application Throughput for Computations at Scale. In *Workshop on Resilience held at the IEEE Intl. Conf. on Cluster Computing and the Grid*, May 2008.

[5] X. Dong, Y. Xie, N. Muralimanohar, and N. P. Jouppi. Hybrid checkpointing using emerging nonvolatile memories for future exascale systems. *ACM Transactions on Architecture and Code Optimization*, 8:6:1–6:29, June 2011.

[6] J. Dongarra and P. Beckman. International Exascale Software Project Roadmap. *International Journal of High Performance Computer Applications*, 25(1), 2011.

[7] A. Geist and R. Lucas. Major computer science challenges at exascale. In *Exascale.org*, Feb. 2009.

[8] G. Grider. ExaScale FSIO: Can we get there? Can we afford to? In *HEC FSIO R&D Workshop*, July 2010.

[9] E. Hendriks. Bproc: the beowulf distributed process space. In *Proc. of the 16th Intl. Conf. on Supercomputing*, pages 129–136. ACM, 2002.

[10] BeoSim Website. http://www.parl.clemson.edu/beosim.

[11] W. M. Jones. Network-aware selective job checkpoint and migration to enhance co-allocation in multi-cluster systems. In *Journal of Concurrency and Computation: Practice and Experience*, volume 21, pages 1672–1691. John Wiley and Sons, Ltd., September 2009.

[12] W. M. Jones, J. T. Daly, and N. A. DeBardeleben. Application resilience: Making progress in spite of failure. In *The Workshop on Resilience held in conjunction with the IEEE Intl. Conf. on Cluster Computing and the Grid*, pages 789–794, May 2008.

[13] W. M. Jones, J. T. Daly, and N. A. DeBardeleben. Impact of sub-optimal checkpoint intervals on application efficiency in computational clusters. In *HPDC '10: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 276–279, 2010.

[14] 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.

[15] ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems. DARPA, 2008.

[16] A. Moody and G. Bronevetsky. Scalable I/O Systems via Node-Local Storage: Approaching 1 TB/sec File I/O. In *Lawrence Livermore National Laboratory: Technical Report #415791*, 2009.

[17] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *Proc. of the ACM/IEEE Intl. Conf. for High Perf. Comp., Networking, Storage and Analysis*, pages 1–11, 2010.

[18] R. A. Ballance et al. Application Monitoring. Cray User Group Meeting, May 2008.

[19] B. Schroeder and G. Gibson. A large-scale study of failures in high-performance computing systems. In *International Conference on Dependable Systems and Networks*, pages 249–258, 2006.

[20] B. Schroeder and G. Gibson. Understanding failures in petascale computers. In *J. of Physics*, 2007.

[21] N. D. Singpurwalla and A. G. Wilson. Probability, chance and the probability of chance. In *IIE Transactions*, volume 41, pages 12–22, Jan 2009.

[22] Vivek Sarkar et al. ExaScale Computing Software Study: Software Challenges in Extreme Scale Systems. DARPA, September 2009.

[23] Ubiquitous High Perf. Comp. (UHPC) Request for Information (RFI). DARPA-SN-09-46, 2009.

[24] J. W. Young. A first-order approximation to the optimum checkpoint interval. In *Communications of the ACM*, pages 530–531, September 1974.