Evaluating Energy Savings for Checkpoint/Restart

Bryan Mills*
University of Pittsburgh
Department of Computer Science
bmills@cs.pitt.edu

Ryan E. Grant and Kurt B. Ferreira†
Sandia National Laboratories
Scalable System Software Department
{regrant|kbferre}@sandia.gov

Rolf Riesen
IBM Research - Ireland
rolf.riesen@ie.ibm.com

ABSTRACT

The U. S. Department of Energy has identified resilience and energy consumption as key challenges for future extreme-scale systems. All checkpoint/restart methods require I/O to local or remote storage. Efforts are under way to minimize the amount of data movement and increase scalability. Nevertheless, the energy consumed by fault resilience methods will increase with system size. It is therefore important to understand the performance overhead in conjunction with the energy consumption of each fault resilience method. In this paper we explore throttling CPU power consumption during I/O intensive checkpoint operations of real applications. We find that 10% total energy savings are possible with little impact on application time to solution.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Fault Tolerance; D.2.8 [**Software Engineering**]: Metrics—*performance measures*

General Terms

Measurement, Performance, Reliability, Power, Energy, Power Saving, Energy Saving, Checkpointing, Fault Tolerance

1. INTRODUCTION

In large capability computing systems, the number of sockets will continue to grow, resulting in decreased reliability and increased energy consumption. These two factors have repeatedly been identified as major challenges along the road to exascale-class systems [35, 28]. With socket-counts on the rise and increased system failure rates [40], future systems will encounter more reliability events. For checkpoint/restore methods, this means more overhead and higher system energy consumption. This makes it imperative to understand the impact fault tolerance methods have on energy consumption.

Fault tolerance in today's large scale production systems relies almost exclusively on coordinated checkpoint/restart. During normal operation, checkpoint/restart (or *rollback recovery*) protocols [9], periodically record the state of all application processes to stable storage (the checkpoint stage). When a process fails, a new incarnation of the failed process is *recovered* from the most recent checkpoint (the restart phase). This limits the amount of lost work to only that since the last checkpoint (the rework stage).

The prevalence of checkpoint/restart is due to a number of factors: failures have, until now, been relatively rare events, applications are generally self-synchronizing, and application state can be saved and restored much more quickly than a given system's mean time to interrupt (MTTI). All of these factors have kept the overheads of traditional checkpoint/restart on current systems limited to a modest portion (currently perhaps 10-25%) of an applications total time to solution. For future extreme-scale systems, a number of these assumptions may change such that the overheads of traditional checkpoint/restart could become prohibitively expensive [29, 10, 39].

Unreliable systems are nothing new. For example, ASCI White originally had a Mean Time Between Failure (MTBF) of only 5 hours [41]. This was later improved to ~50 hours, but exascale-class systems will observe failure rates significantly higher than even poorly behaving HPC systems of the past. Worse than that, the methods to alleviate the impact of failures on system performance, not only add overhead, but also increase power consumption. Figure 1 exemplifies the problem: The time to checkpoint and restart increases exponentially with system size. These are high-power operations and, therefore, energy consumption will also increase at a much higher rate than the increase in number of sockets would suggest.

A number of recent studies show that general power consumption can be reduced during writing of a checkpoint to stable storage [36, 25]. The CPU is the largest consumer of power on an HPC node, but its power consumption can be controlled using Dynamic Voltage and Frequency Scaling (DVFS). The prior work suggests that during the I/O intensive checkpoint and restart operations, throttling the CPU

*This work is supported in part by NSF grants CNS12-53218 and CNS12-52306

†Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

E2SC '13 November 17 - 21 2013, Denver, CO, USA

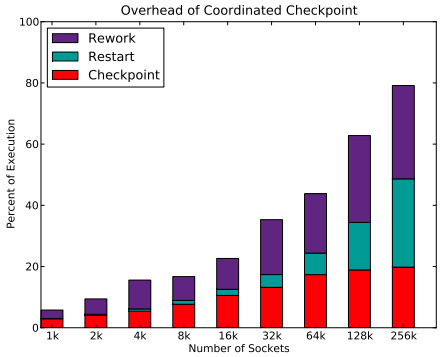


Figure 1: Percent of wall-clock time spent in each coordinated checkpointing component using a validated simulator [33]. Checkpoint commit time is 15 minutes, a value shown seen on many extreme-scale systems [4, 11]; and a node MTBF of 15 years [39], using the optimal checkpoint interval from Daly [6].

can save power without impacting checkpoint performance.

In this work, we use a previously published [18] framework for examining component-level power consumption to measure the energy cost of checkpoint operations to local SSD's and remote storage. For remote operations we compare the power consumption of IP and RDMA, both over an InfiniBand network. With these baselines in place, we then use DVFS to throttle CPU speed during checkpoint writes to measure the energy savings and performance impact. We find that overall energy savings of 10% are possible with actual HPC workloads. Furthermore, the choice of network protocol and local versus remote storage have important energy consumption impacts that needs to be considered when designing fault tolerance protocols that make use of hierarchical storage.

2. EXPERIMENTAL RESULTS

Before we present results, we explain our experimental methodology and the framework we used. We then measure the effect of lowering the frequency and voltage of the CPU during a checkpoint write operation to local and remote storage. For remote I/O over InfiniBand, we switch between the IP and the RDMA protocol and evaluate each. We then repeat these three experiments for two applications and analyze the results.

2.1 Methodology

We used DVFS to vary the CPU frequency and voltage during a checkpoint write to determine the potential energy savings available. Using DVFS, several different discrete "gears" are available for CPU frequency; we explored all those available on the testbed hardware. We gathered component-level power consumption data from several nodes, and did this while checkpointing to local and remote storage. For remote access, we used two NFS solutions: one using the kernel network stack and the other using the IB RDMA interface.

2.2 Experimental Framework

We measured power consumption on a cluster with 104 nodes, each equipped with an AMD Llano Fusion APU, which is a 4-core AMD K10 x86 paired with a 400-core Radeon HD 6550D. For our experiments we only used the x86 cores, ignoring the available GPU. The CPU frequency and voltage are modified using the powernow_k8 kernel module. There are six available gears ranging from a frequency of 3.8 GHz down to 1.4 GHz.

Component level power measurements of the CPU, memory, on-node SSD device, motherboard and the Qlogic QDR InfiniBand HCA were performed using a custom designed power measurement system. More detail about the system is available in [18].

We used LAMMPS [32], a molecular dynamics code, and HPCCG, a conjugate gradient solver from Sandia's mantevo suite [38] as the MPI applications for our experiments. Both are important as they represent a range of computational techniques. LAMMPS is a production level code that is frequently run at very large scales on U. S. Department of Energy leadership class systems, while HPCCG is a mini-app that is representative of a real, finite element code. Both used Open MPI and the built-in BLCR [14] support for checkpointing.

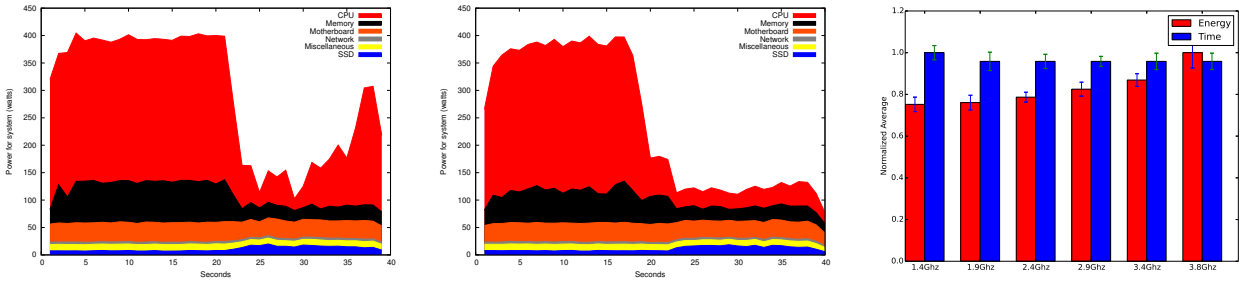
2.3 Local Checkpoint Power Profile

Checkpointing is an I/O intensive operation and previous work has indicated that CPU utilization is low during a local checkpoint [36]. This section explores what energy savings may be possible when checkpointing to a local SSD. Figure 2(a) shows the component level power profile of 4 nodes running at full processor speed performing a local coordinated checkpoint. As the processes pause their execution there is a drop in the amount of power consumed by the CPU, even without modifying the operating voltage or frequency of the processor. This initial result indicates that there is an opportunity to save energy by reducing the clock frequency and voltage of the processor.

Observe in Figure 2(a) that even though CPU power consumption drops during a checkpoint, the CPU is still the dominant consumer of power. All other components consume less than half the power consumed by the CPU. This makes CPU power an excellent candidate for energy savings during checkpoints. SSD power consumption does increase during the checkpoint, but it is not as promising a candidate for energy savings as the CPU as the percentage of energy consumed by the SSD is far less than that of the CPU.

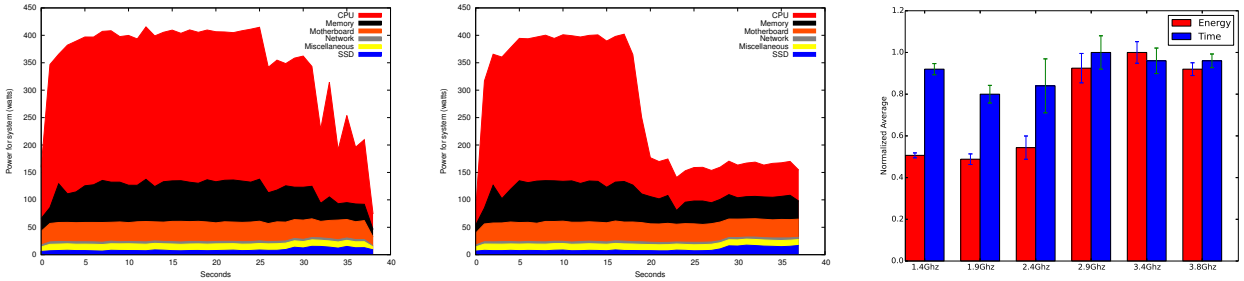
Figure 2(b) shows the result of checkpointing with the processor frequency set to 1.4 GHz, the lowest possible gear. Reducing the CPU frequency (and voltage) reduces total energy consumption and smooths out power consumption during the checkpoint time. During the I/O operation, there are times that the CPU is blocked in a busy loop waiting for the I/O to complete. This polling for I/O completion can lead to small power spikes. By reducing the CPU frequency, these spikes occur less frequently because the CPU is less likely to block on I/O.

Reducing the CPU frequency during checkpoint causes the operation to take slightly longer. In Figure 2(c) we compare the consumed energy during the checkpoint time and the total execution time for the operation. We show the results for all 6 available voltage and frequency gears in our environment. To compare both time and energy in the same figure, we normalized the values to the highest measured. This figure confirms that during extended local I/O opera-



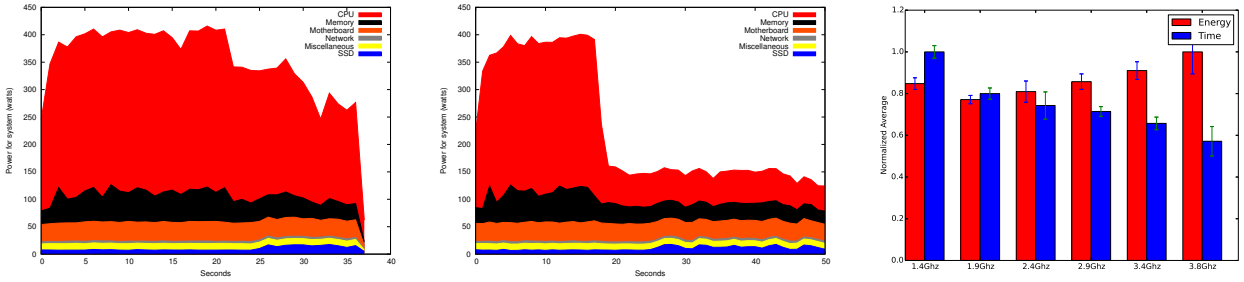
(a) 3.8 GHz (b) 1.4 GHz (c) Time vs Energy

Figure 2: ***Local SSD.*** **Component level power profile during a coordinated checkpoint of HPCCG in a 4-node cluster using 16 processes, each process checkpoint was approximately 1.5GB. The Time versus Energy plot shows the energy and time to complete the checkpoint operation over 10 separate runs. Error bars are the standard deviation.**



(a) 3.8 GHz (b) 1.4 GHz (c) Time vs Energy

Figure 3: ***Remote SSD using IP over InfiniBand.*** **Component level power profile during a coordinated checkpoint of HPCCG in a 4-node cluster using 16 processes, each process checkpoint was approximately 1.5GB. The Time versus Energy plot shows the energy and time to complete the checkpoint operation over 10 separate runs. Error bars are the standard deviation.**



(a) 3.8 GHz (b) 1.4 GHz (c) Time vs Energy

Figure 4: ***Remote SSD using RDMA over InfiniBand.*** **Component level power profile during a coordinated checkpoint of HPCCG in a 4-node cluster using 16 processes, each process checkpoint was approximately 1.5GB. The Time versus Energy plot shows the energy and time to complete the checkpoint operation over 10 separate runs, error bars are the standard deviation.**

tions, reducing the CPU power has little effect on the time to completion, but can save up to 25% of the total energy.

These results are very encouraging, but for a checkpoint to be usable it must be stored on a device that is failure independent of the node performing the computation. In practice this means that writing a checkpoint also includes a network operation, to copy or stream the checkpoint data to a network storage device. The next section will explore the power profile of writing a checkpoint to a network device.

2.4 Power Consumption by Network Type

Checkpoints are clearly I/O bound operations, and checkpointing to a remote location will therefore be network intensive. CPU involvement in the network operations is highly dependent upon both the software and hardware being utilized. If network operations require significant CPU resources, reducing processor frequency can have a significant effect on the time necessary to write a checkpoint. This will impact the potential energy savings for checkpointing over a network. In order to study the effect of distributing checkpoints across a network we chose to store checkpoints in neighboring compute nodes. Because this is a coordinated checkpoint there should be no interference with the MPI application being checkpointed. We tested two different network configurations using NFS: IP over InfiniBand and RDMA over InfiniBand.

Both of the network configurations tested use InfiniBand network hardware. InfiniBand networks can be implemented as an offloaded or onloaded, or partially onloaded and offloaded solution. This paper examines the energy consumption of a system with a partially onloaded InfiniBand Qlogic host channel adapter (HCA). With a fully offloaded InfiniBand HCA, such as those from Mellanox, the CPU utilization during the checkpoint could reasonably be expected to be lower, and therefore greater savings may be possible.

2.4.1 IP over InfiniBand

IP over InfiniBand (IPoIB) is a protocol that allows encapsulating IP packets for their transmission over InfiniBand network hardware [5]. This requires mapping IP addresses to InfiniBand subnets that support IPoIB. The underlying network driver/hardware is an InfiniBand HCA, which transmits the encapsulated packets inside of native InfiniBand messages. IPoIB utilizes portions of the kernel IP networking stack, and associated upper layer transports (e.g. TCP/UDP). Therefore, the performance benefits of OS bypass is not available to IPoIB applications and CPU load is increased over native IB. IPoIB therefore provides the convenience of a socket interface to an application but with the drawback of a performance penalty.

2.4.2 RDMA over InfiniBand

Remote Direct Memory Access (RDMA) is a key feature of InfiniBand networks. It allows for a source node to transmit data directly into a target node's memory. There are two methods for performing RDMA. The send/recv method uses target side "recv" queue (RQ) entries that are matched to incoming messages. These RQ entries indicate where a given message should be placed in memory, which can be an application's buffer, avoiding any intermediary copies that would otherwise be performed in a typical kernel network transport communication.

The other method of performing RDMA is the Write/Read approach, which has the source node include all of the information on where the data is to be placed in the target node's memory. This requires that the source node has knowledge of the target node's memory, including what areas are designated for that source node's messages. This is typically accomplished through an exchange of data prior to RDMA communication, or through buffer advertisement while communication is ongoing.

RDMA can provide very low latency networking, and small message RDMA operations can have sub-microsecond latencies, while large messages can have very high throughput.

2.4.3 Remote Checkpoint Power Profile

Figure 3 shows the power profile of writing a checkpoint over a network using IP over InfiniBand. When executing the checkpoint at full speed there is significantly more CPU activity than that observed for local SSD checkpoints (Figure 2). This increased CPU utilization is due to the network stack processing required by our onloaded InfiniBand hardware. Reducing the CPU speed reduces the energy consumption significantly while even potentially increasing checkpoint performance. This result is encouraging and shows that reducing CPU power can result in energy savings with little additional overhead, and in the case where resource contention was causing slowdown, actually increase checkpoint efficiency.

It would be reasonable to expect that a local SSD checkpoint would be faster than a network operation, however we consistently saw full speed IP over InfiniBand and RDMA outperform local SSD writes, albeit only by a small amount. The reason for this unexpected result is the buffering behavior of NFS. Due to it's write buffers, NFS reports to the client that the write operation is complete as soon as the entire message has been buffered at the server. The actual write to disk then finishes, allowing the client to proceed with computation. This results in the network copy appearing to be slightly faster than the local SSD write, as local SSD write-caching is not available in the Linux kernel we used for testing. As can be observed in 3(c), reducing the CPU power during checkpoint operation can save 50% of the consumed energy. Further research is necessary to determine the impact that the NFS buffered writes might have upon energy and time to solution.

In contrast, Figure 4(a) shows a near constant use of the CPU during the RDMA network transfer. Although, in principle, RDMA is OS bypass and can be offloaded, our interface cards make heavy use of the CPU during RDMA transmission. With an un-throttled CPU, the transmission is slightly faster than IPoIB. Reducing the CPU speed we observe that RDMA takes significantly longer than it did at 3.8 GHz. Figure 4(c) shows that while we can save 15% of the energy, this causes the checkpoint time to nearly double.

2.5 Checkpoint Energy over Application Execution

In the previous sections we examined the power profile of a single checkpoint event during the execution of HPCCG. In this section, we look at the power profile over three checkpoints taken during a run of LAMMPS using the Lennard-Jones workload.

Figure 6(a) shows the power profile of LAMMPS when the three checkpoints go to the local SSD. During the local checkpoints, the CPU power consumption is considerably

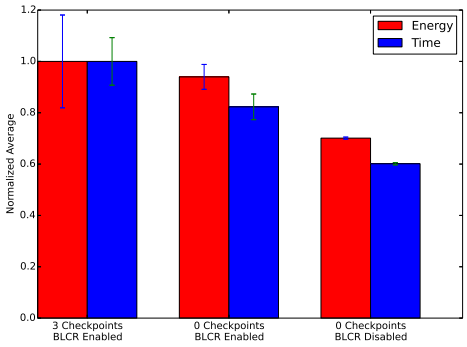


Figure 5: Time to solution and energy consumed for LAMMPS using different configurations. These experiments were ran using Open MPI 1.3.4.

reduced. However, when writing remote checkpoints, shown in Figure 7(a) and 8(a), the CPU power is much higher during these times. With reduced CPU frequency shown in Figures 6(b), 7(b) and 8(b), we see a drop in power consumption and an increase in execution time, particularly for the off-node operations.

By focusing on the checkpoint event itself in previous experiments, we were able to draw some conclusions regarding the use of DVFS during those events. However, when evaluating the energy and time to solution we found that the variance of the application runtime was too high to draw any conclusions. This was unexpected as LAMMPS typically has very little variance in execution time. Further investigation determined that the variance was introduced by a helper thread for Open MPI's BLCR support. Figure 5 shows the effect BLCR support in Open MPI has on LAMMPS runtimes even when no checkpoints are taken.

The amount of variance is further magnified when checkpoints are actually written. This is because the time to coordinate the checkpoint is dependent upon how far into the application execution it is requested. The variance introduced by Open MPI BLCR support moves this request from run to run and obscures conclusions. Therefore, we do not show energy graphs for the overall application run.

We can, however, conclude from the power profiles that the behavior during the checkpoint event is the same even when executing multiple checkpoints over the execution of the application. Because these profiles show the similar behavior to that found in the HPCCG experiments, this should translate to energy savings in the overall application execution. Future work will need to address the variance introduced by the checkpointing support in Open MPI to be able to confirm this conclusion.

3. RELATED WORK

DVFS is a mechanism by which the frequency and voltage of a processor can be scaled during CPU operation. Power can be approximated by $P = \alpha CV^2 f$, where α is the activity factor of the CPU, C is the capacitance, V the voltage and f the frequency. The reduction in both the frequency and voltage of a processor has a cubic relation to the amount of power (instantaneous energy consumption) of a CPU. DVFS [30] and clock throttling [26] are leveraged by energy saving techniques since CPU power currently dominates overall system power consumption [15].

Power awareness is a mature research area and several energy saving runtime techniques [13, 15, 16, 22, 23, 24, 34, 43] have been proposed, mostly concentrating on portions of execution that exhibit enough slack to take advantage of a lower CPU operating frequency. These are runtime methods and use whole system activity via performance monitoring counters (PMCs) or specific communication middleware (e.g. MPI) calls to identify periods of slack. While these methods *may* be able to adapt to I/O phases during resilience events, they are not specifically designed to take advantage of the explicit notification that such an event is starting or ending. For adaptive methods, this leads to delays between the event occurring and its identification. This leads to energy inefficiency at the beginning of a checkpoint operation and computational inefficiency at the end of a checkpoint.

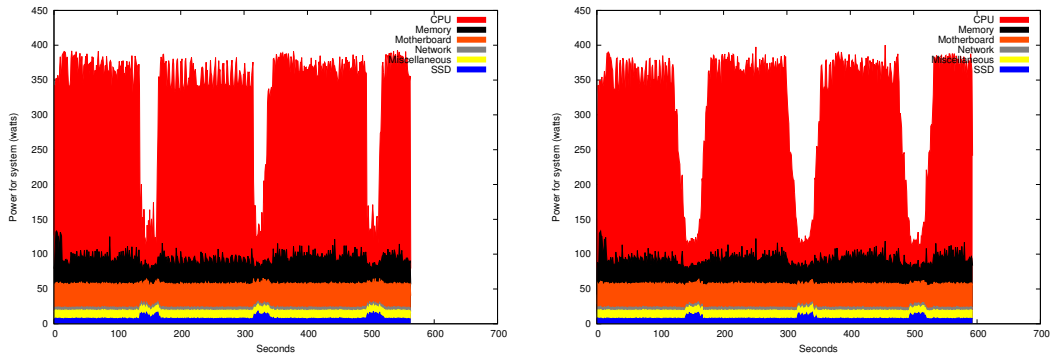
The energy consumption of systems during checkpoints was studied in [7] for a variety of checkpointing methods. Models have been developed [25] for coordinated, uncoordinated (message logging) and parallel recovery methods. However, none of the assessments of the energy consumption of checkpoints to remote storage have considered CPU frequency scaling during checkpoints. Therefore, while the power profile of several systems is understood during checkpointing, the potential energy savings of DVFS and the corresponding performance penalties are unknown.

The energy efficiency of *local* I/O for checkpoint operations has been assessed, and a scheme for saving energy during local I/O operations using DVFS has been proposed in [36]. The energy efficiency and the corresponding impact on performance shown in [36] is encouraging and confirmed in this paper. We go one step further and also evaluate DVFS power throttling during *remote* I/O operations.

Prior work has only provided whole system energy/power measurement, while this paper explores the energy/power characteristics of individual system components during the checkpoint. This allows for a more comprehensive analysis of the system behavior during checkpointing. In addition, all of the work on checkpoint energy to date has used sampling equipment with a 1 second sampling period. This paper uses a higher sampling rate, and the samples were taken from the output side of the power supply (not between the power supply and the wall plug). The in-system measurement demonstrates the energy/power profile without the variation that may be caused due to different power supply efficiencies.

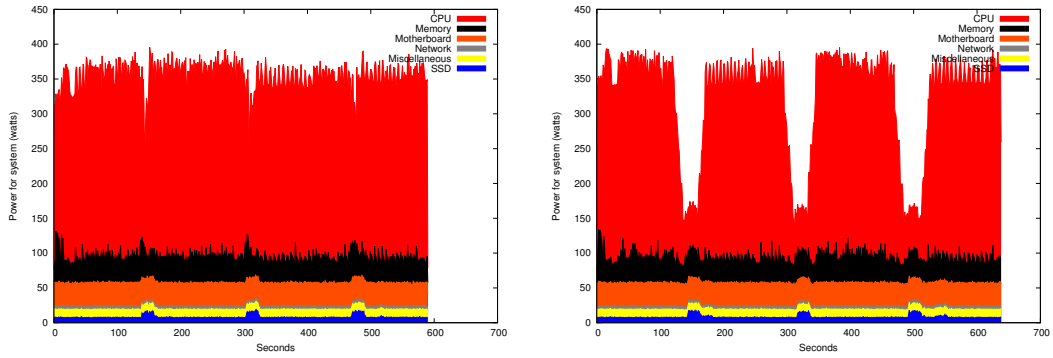
There has been a great effort in the community to optimize rollback/recovery protocols. These optimizations take a number of forms: from committing checkpoints to high-bandwidth stable storage (e.g. NVRAM [21, 8]), methods which decrease the time to write each individual checkpoint (e.g. incremental checkpointing [3, 37, 1, 12], multi-level checkpointing [44, 31, 27], remote checkpointing [42, 45], and checkpoint compression [17]), and methods that decrease the number of checkpoints that must be taken per unit time (e.g. replication [10]). With these advances, there is a strong belief that that checkpoint/restart methods will continue to be viable for future extreme-class platforms. Consequently, saving energy during rollback/recovery operations is of great interest to the research community.

In addition to coordinated checkpoint/restart, uncoordinated or asynchronous checkpointing [2, 19, 20] has been suggested as an alternative resilience mechanism. Unco-



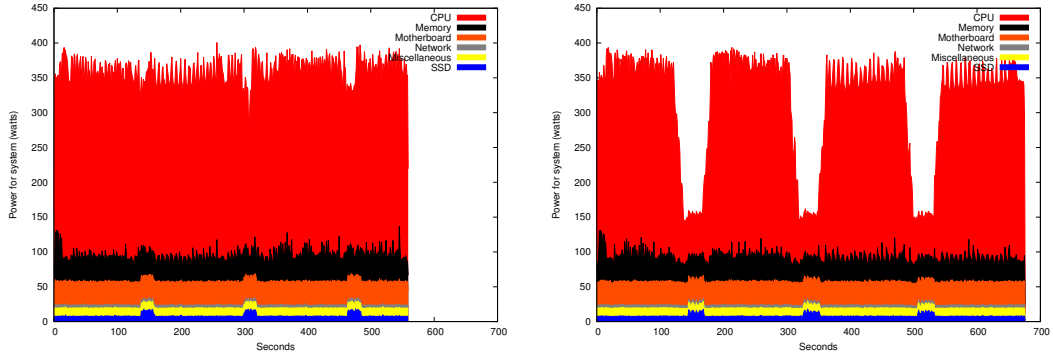
(a) 3.8 Ghz (b) 1.4 Ghz

Figure 6: *Local SSD.* Component level power profile during three coordinated checkpoints of a LAMMPS application run within a 4-node cluster using 16 processes, each process checkpoint was approximately 700MB.



(a) 3.8 Ghz (b) 1.4 Ghz

Figure 7: *Remote SSD using IP over InfiniBand.* Component level power profile during three coordinated checkpoints of a LAMMPS application run within a 4-node cluster using 16 processes, each process checkpoint was approximately 700MB.



(a) 3.8 Ghz (b) 1.4 Ghz

Figure 8: *Remote SSD using RDMA over InfiniBand.* Component level power profile during three coordinated checkpoints of a LAMMPS application run within a 4-node cluster using 16 processes, each process checkpoint was approximately 700MB.

ordinated checkpointing allows each process to checkpoint independently, thereby avoiding synchronization overheads and reducing I/O contention. Uncoordinated checkpointing protocols do not require non-failed nodes to rollback. Methods used in this work to optimize energy consumption hold relevance to uncoordinated checkpointing as well.

4. CONCLUSIONS AND FUTURE WORK

This work demonstrates there is potential for realizing energy reduction during checkpointing events using DVFS –

all while having little to no impact on checkpointing performance. More specifically, we show a maximum 50% energy savings by throttling CPU power consumption during I/O intensive checkpoint operations. Given that these checkpoint operations can consume 20% of an applications total runtime (see Figure 1), this leads to a possible 10% total application energy savings from DVFS with checkpointing. We also show that this potential is highly dependent upon the network characteristics. For the Qlogic InfiniBand cards in our test cluster, the opportunity to save energy is small compared to the benefits seen committing checkpoints to local storage due to network onload versus offload issues. The onloaded interface used showed IPoIB, while slower than RDMA, has the greater potential to save energy during large I/O operations. This result is in contrast to general observations in existing literature, as this is the first of its kind to explore DVFS techniques using IP over InfiniBand and RDMA during checkpoint operations.

Understanding the potential energy savings during checkpointing periods as well as the interplay between CPU performance and checkpoint commit speed provides the building blocks for future power-aware checkpointing research. In a broader scope, this work demonstrates that checkpoint events can involve significant amounts of CPU usage depending on the system configuration. We believe this finding could have potential impact on the performance of recently suggested staged checkpoints, in which checkpoints are written locally then copied over the network while the application continues to execute. If one has a system in which the CPU is heavily involved, the copy operation will be slower than expected and will likely interfere significantly with application progress.

Going forward, we will explore energy savings using a fully offloaded InfiniBand network card. We believe the energy savings possible will more closely match those found in the local checkpoint case. We also plan on exploring the use of more traditional parallel file systems in addition to our local storage system presented in this paper. Additionally, we are exploring energy optimizations in other parts of rollback/recovery; for example, the restart and rework phases. Lastly, we plan to explore different checkpoint methods, including uncoordinated checkpointing. Our hypothesis is that uncoordinated checkpointing will benefit from these techniques. However, due to the lack of coordination, the performance implications are not clear.

5. REFERENCES

[1] S. Agarwal, R. Garg, M. S. Gupta, and J. E. Moreira. Adaptive incremental checkpointing for massively parallel systems. In *Proceedings of the 2004 International Conference on Supercomputing*, St. Malo, France, 2004.

[2] J. Ahn. 2-step algorithm for enhancing effectiveness of sender-based message logging. In *SpringSim '07: Proceedings of the 2007 spring simulation multiconference*, pages 429–434, 2007.

[3] G. Bronevetsky, D. Marques, K. Pingali, S. McKee, and R. Rugina. Compiler-enhanced incremental checkpointing for openmp applications. In *IEEE International Symposium on Parallel&Distributed Processing*, pages 1–12, 2009.

[4] F. Cappello. Fault tolerance in petascale/ exascale systems: Current knowledge, challenges and research opportunities. *IJHPCA*, 23(3):212–226, 2009.

[5] J. Chu and V. Kashyap. Transmission of ip over infiniband (ipoib). Technical report, RFC 4391, April, 2006.

[6] J. T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Gener. Comput. Syst.*, 22(3):303–312, 2006.

[7] M. Diouri, O. Gluck, L. Lefèvre, and F. Cappello. Energy considerations in checkpointing and fault tolerance protocols. In *Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on*, pages 1–6. IEEE, 2012.

[8] X. Dong, N. Muralimanohar, N. Jouppi, R. Kaufmann, and Y. Xie. Leveraging 3d pcram technologies to reduce checkpoint overhead for future exascale systems. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 57:1–57:12, New York, NY, USA, 2009. ACM.

[9] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3):375–408, 2002.

[10] K. Ferreira, R. Riesen, P. Bridges, D. Arnold, J. Stearley, J. H. L. III, R. Oldfield, K. Pedretti, and R. Brightwell. Evaluating the viability of process replication reliability for exascale systems. In *SC*, Nov. 2011.

[11] K. B. Ferreira. *Keeping Checkpoint/Restart Viable for Exascale Systems*. PhD thesis, University of New Mexico, Department of Computer Science, Dec. 2011.

[12] K. B. Ferreira, R. Riesen, R. Brightwell, P. G. Bridges, and D. Arnold. Libhashckpt: Hash-based incremental checkpointing using GPUs. In *Proceedings of the 18th EuroMPI Conference*, Santorini, Greece, September 2011.

[13] R. Ge, X. Feng, W.-c. Feng, and K. W. Cameron. CPU miser: A performance-directed, run-time system for power-aware clusters. In *International Conference on Parallel Processing (ICPP)*, pages 18–18. IEEE, 2007.

[14] P. H. Hargrove and J. C. Duell. Berkeley lab checkpoint/restart (blcr) for linux clusters. In *Journal of Physics: Conference Series*, volume 46, page 494. IOP Publishing, 2006.

[15] C.-h. Hsu and W.-c. Feng. A power-aware run-time system for high-performance computing. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 1. IEEE Computer Society, 2005.

[16] S. Huang and W. Feng. Energy-efficient cluster computing via accurate workload characterization. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 68–75. IEEE Computer Society, 2009.

[17] D. Ibtesham, D. Arnold, P. G. Bridges, K. B. Ferreira, and R. Brightwell. On the viability of compression for reducing the overheads of checkpoint/restart-based fault tolerance. *2012 41st International Conference on Parallel Processing*, 0:148–157, 2012.

[18] J. H. L. III, P. Pokorny, and D. DeBonis. Powerinsight

- a commodity power measurement capability. In *The Third International Workshop on Power Measurement and Profiling in conjunction with IEEE IGCC 2013*, Arlington Va, 2013.

[19] Q. Jiang and D. Manivannan. An optimistic checkpointing and selective approach for consistent global checkpoint collection in distributed systems. In *Proceedings of the 2007 IEEE International Parallel and Distributed Processing Symposium*, Mar. 2007.

[20] D. B. Johnson and W. Zwaenepoel. Recovery in distributed systems using asynchronous and checkpointing. In *Proceedings of the seventh annual ACM Symposium on Principles of distributed computing*, pages 171–181, 1988.

[21] S. Kannan, A. Gavrilovska, K. Schwan, and D. Milojicic. Optimizing checkpoints using nvm as virtual memory. In *Proceedings of the nternational Parallel and Distributed Processing Symposium*, IPDPS '13, New York, NY,USA, 2013. ACM.

[22] N. Kappiah, V. W. Freeh, and D. K. Lowenthal. Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in MPI programs. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 33. IEEE Computer Society, 2005.

[23] D. Li, B. de Supinski, M. Schulz, D. Nikolopoulos, and K. Cameron. Strategies for energy efficient resource management of hybrid programming models. *Parallel and Distributed Systems, IEEE Transactions on*, 24(1):144–157, 2013.

[24] M. Y. Lim, V. W. Freeh, and D. K. Lowenthal. Adaptive, transparent frequency and voltage scaling of communication phases in MPI programs. In *SC 2006 Conference, Proceedings of the ACM/IEEE*, pages 14–14. IEEE, 2006.

[25] E. Meneses, O. Sarood, and L. V. Kale. Assessing energy efficiency of fault tolerance protocols for HPC systems. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium on*, pages 35–42. IEEE, 2012.

[26] M. Mittal and R. Valentine. Performance throttling to reduce IC power consumption, Feb. 17 1998. US Patent 5,719,800.

[27] A. Moody, G. Bronevetsky, K. Mohror, and B. R. de Supinski. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC '10)*, pages 1–11, 2010.

[28] U. D. of Energy Office of Science. The opportunities and challenges of exascale computing, 2010.

[29] R. A. Oldfield, S. Arunagiri, P. J. Teller, S. Seelam, M. R. Varela, R. Riesen, and P. C. Roth. Modeling the impact of checkpoints on next-generation systems. In *24th IEEE Conference on Mass Storage Systems and Technologies*, pages 30–46, Sept. 2007.

[30] T. Pering, T. Burd, and R. Brodersen. Dynamic voltage scaling and the design of a low-power microprocessor system. In *Power Driven Microarchitecture Workshop, attached to ISCA98*, pages 96–101, 1998.

[31] J. Plank, K. Li, and M. Puening. Diskless checkpointing. *Parallel and Distributed Systems, IEEE Transactions on*, 9(10):972–986, oct 1998.

[32] S. J. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal Computation Physics*, 117, 1995.

[33] Rolf Riesen, Kurt Ferreira, Jon Stearley, Ron Oldfield, James H. Laros III, Kevin Pedretti and Ron Brightwell. *Redundant Computing for Exascale Systems*. Sandia National Laboratories, December 2010. Sandia Report SAND2010-8709.

[34] B. Rountree, D. K. Lownenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch. Adagio: making DVS practical for complex HPC applications. In *Proceedings of the 23rd international conference on Supercomputing*, pages 460–469. ACM, 2009.

[35] S. R. Sachs. Tools for exascale computing: Challenges and strategies, 2011.

[36] T. Saito, K. Sato, H. Sato, and S. Matsuoka. Energy-aware I/O optimization for checkpoint and restart on a NAND flash memory system. In *Proceedings of the 3rd Workshop on Fault-tolerance for HPC at extreme scale*, pages 41–48. ACM, 2013.

[37] J. C. Sancho, F. Petrini, G. Johnson, J. Fernandez, and E. Frachtenberg. On the feasibility of incremental checkpointing for scientific computing. In *Proceedings of the 2004 International Parallel and Distributed Processing Symposium*, Santa Fe, New Mexico USA, 2004.

[38] Sandia National Laboratory. Mantevo project home page, 2010.

[39] B. Schroeder and G. A. Gibson. A large-scale study of failures in high-performance computing systems. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN2006)*, June 2006.

[40] B. Schroeder and G. A. Gibson. A large-scale study of failures in high-performance computing systems. *Dependable and Secure Computing, IEEE Transactions on*, 7(4):337–350, 2010.

[41] M. Seager. Operational machines: ASCI white. In *7th Workshop on Distributed Supercomputing, Durango, CO*, 2003.

[42] G. Stellner. Cocheck: Checkpointing and process migration for MPI. In *International Parallel Processing Symposium*, pages 526–531, Honolulu, HI, April 1996. IEEE Computer Society.

[43] A. Tiwari, M. Laurenzano, J. Peraza, L. Carrington, and A. Snavely. Green queue: Customized large-scale clock frequency scaling. In *2012 Second International Conference on Cloud and Green Computing (CGC)*, pages 260–267. IEEE, 2012.

[44] N. H. Vaidya. A case for two-level distributed recovery schemes. In *ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '95/PERFORMANCE '95, pages 64–73, New York, NY, USA, 1995. ACM.

[45] V. C. Zandy, B. P. Miller, and M. Livny. Process hijacking. In *8th International Symposium on High Performance Distributed Computing (HPDC '99)*, pages 177–184, Redondo Beach, CA, August 1999.