SAND2014-0696C

Metrics for Evaluating Energy Saving Techniques for Resilient HPC Systems

Ryan E. Grant, Stephen L. Olivier, Jim H. Laros III, Ron Brightwell
Sandia National Laboratories*
P.O. Box 5800, MS-1110
Albuquerque, NM 87185-1110
Email: {regrant, slolivi, jhlaros, rbbrigh}@sandia.gov

Allan K. Porterfield
RENCI
100 Europa Dr, Ste 540
Chapel Hill, NC 27517
Email: akp@renci.org

Abstract—**The metrics used for evaluating energy saving techniques for future HPC systems are critical to the correct assessment of proposed methods. Current predictions forecast that overcoming reduced system reliability, increased power requirements and energy consumption will be a major design challenge for future systems. Modern runtime energy-saving research efforts do not take into account the energy spent providing reliability. They also do not account for the increase in the probability of failure during application execution due to runtime overhead from energy saving methods. While this is very reasonable for current systems, it is insufficient for future generation systems. By taking into account the energy consumption ramifications of increased runtimes on system reliability, better energy saving techniques can be developed. This paper demonstrates how to determine the impact of runtime energy conservation methods within the context of failure-prone large scale systems. In addition, a survey of several energy savings methodologies is conducted and an analysis is performed with respect to their effectiveness in an environment in which failures occur.**

Keywords*—*energy saving, HPC, reliability, power, DVFS, frequency scaling, voltage scaling

I. Introduction

The ever increasing size of large computational systems has corresponded to increases in power/energy consumption and decreases in the overall reliability of computing platforms. Current energy consumption metrics are insufficient to describe the potential of proposed energy saving techniques. This paper demonstrates new methods of determining the energy savings of proposed techniques on large systems in the context of unreliable hardware.

Reliability has been a concern for a long time, and systems that had poor Mean Time Between Failure (MTBF) have existed in the past. For example, in 2001, ASCI white had a MTBF of 5 hours [1] (later improved to 50 hours). Future systems are forecast to have a much lower system MTBF than current hardware. Therefore a MTBF of less than 5 hours is possible. As application runs of several hours or longer are common for HPC, even runtime increases of 10% or less (a reasonable but not desirable, upper bound on performance degradation for energy saving) could be expected to impact the probability that a failure occurs during an application's runtime.

Energy saving techniques can increase MTBF by providing lower temperatures during runtime. However, the increases that can be obtained by using lower power approaches must be greater than the performance impact on runtime in order to offset the likelihood that a failure occurs during the additional runtime imposed by the energy saving technique overhead.

The lack of an easy to use model for quantifying the effect of energy saving techniques on system failure, as well as a history that includes relatively good MTBFs for deployed systems has prevented the adoption of reliability concerns into the metrics used to assess runtime energy saving techniques. It has been shown that the number of system failures increases linearly with socket count [2], and consequently, the probability that a failure occurs at any given point in time also scales with node count. Unfortunately, the runtime of an application does not scale perfectly with increasing node counts. Therefore, when determining the energy savings potential of a new runtime system, one simply cannot obtain a runtime, divide by a number of nodes/sockets greater than those tested and calculate an energy efficiency given a linear scaling of failure events. However, a calculation based on the slope of the scaling curve for a limited number of sockets can provide a best case energy-reliability number that can be used to assess different techniques.

A comparison between energy savings techniques that takes into account reliability mechanisms is a more accurate way to evaluate existing energy saving schemes for future large-scale systems. The trade-offs between increased runtime and reduced energy consumption are not always clearly observable. An energy-reliability metric provides an easier way to compare between proposed techniques. Energy-reliability imposes a quantifiable penalty to energy savings techniques for increased runtime, instead of relying on "user acceptance" arguments. For example, given two techniques, one of which provides an average energy savings of 5% and a runtime increase of 2% and one that provides savings of 7% and runtime increase of 4%, which is the better technique? If we consider the "user acceptance" model, then both could be acceptable solutions to a user, as their runtime increases are not excessively large. Therefore, one could reasonably assume that the second technique is superior. However, if the second technique is applied to a long application run on a large system, the increased runtime of +2% over the first technique may correspond to

*Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

it encountering an additional system failure and subsequent restore that may completely mitigate any energy savings and in fact use more energy than the first technique.

This paper seeks to provide a new metric for energy-saving research which will allow comparison of energy saving methods in unreliable systems and provides the tools to come to quantifiable conclusions as to usefulness of energy saving techniques for varying system sizes. This will help to identify techniques that may have otherwise been discarded due to low average energy savings but that have very little overhead. Such techniques may be useful in large supercomputers.

These new metrics are also of use to reliability researchers as they can directly compare their reliability methods in an energy consumption context. For example, if a method can increase the speed of a reliability mechanism (e.g. a faster checkpointing mechanism), without increasing the energy consumed by that mechanism, then the net result is not just the same energy per application run, but an improved energy as the likelihood of a failure during program execution will decrease (due to a lower runtime).

II. Background

Researchers have been concerned with the reliability of systems for many years. Large scale studies of failure rates [2] and the causes of these failures have been performed. Energy consumption is also a well studied area, with several energy saving runtime techniques [3]–[8] having been proposed and implemented. Energy saving techniques such as DVFS [9] and clock throttling [10] have been used to provide power state governors for operating systems. While most studies concentrate on symmetric multiprocessor systems, it is probable that future systems will be more heterogeneous in their processor clock rates if not core architectures. Such systems have been studied [11] and the energy consumption of such systems have been examined [12].

While many energy saving techniques have concentrated on mobile devices, the energy consumption of supercomputers is of great concern. There are two areas in which energy and power for supercomputing is an issue. The first is energy consumption, which directly equates to the cost of running the system. The second is peak power usage, which can constrain a system from running in certain high power states due to limits on the rate at which energy can be consumed.

This proposal concentrates on providing metrics for energy consumption of supercomputers in the presence of failures. While peak power is a concern, the metrics for determining it are well laid out and can be easily leveraged by researchers. The current state of the art in energy consumption is the use of the total number of Joules (J) of energy used for a given test. This can be directly compared to the number of Joules used for a baseline run without energy saving techniques. Typically, this is contrasted by an accounting of the overhead imposed by using the energy saving technique. By definition, the energy saving technique should have some overhead as it is compared to the system running at full speed. These are typically combined in an energy delay (ED) product, or an ED^2 product.

Unfortunately, many power/energy researchers do not have access to large scale power instrumented systems, and therefore cannot easily account for the real world behavior of system failures. Even if such failures did occur during application runs, there would be no practical way to compare them to other results unless a very large number of results were gathered over a long time period, or artificial failures were injected during runtime. To the authors' knowledge, there is no existing standard framework for estimating the impact of failures on the energy consumption of a system for energy saving techniques.

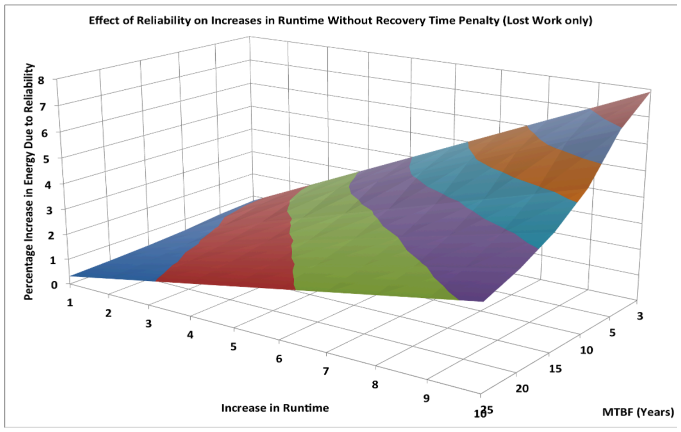


Fig. 1. Energy overhead due to reliability for future systems, assuming no recovery overhead

III. Motivation

Until recently, the reliability of HPC systems has not been a major component in the overall energy usage of an application as errors are relatively infrequent. In future environments where errors may be more frequent, the effect of these reliability events on systems must be taken into account when examining any technique with associated runtime overhead. Figure 1, illustrates the impact that reliability may have on runtime overheads in terms of additional energy usage due to failures. Figure 1 does not take into account any overhead associated with the actual recovery (real methods should take some amount of time, unless they are completely redundant methods like Double Modular Redundancy (DMR) and Triple Modular Redundancy (TMR)). This assumes an amount of lost work based on current checkpoint/recovery schemes and therefore one could expect that the results would be more favorable with more frequent checkpointing intervals. However, real reliability methods will also incur some overhead, lessening any improvements seen by more frequent checkpointing. It should also be noted that more frequent checkpointing will also incur runtime overhead which in turn will increase energy consumption.

IV. Theory

This section outlines the building blocks used to create an energy-reliability metric for use in comparing energy saving techniques for large systems. It derives the equations used for determining energy reliability from common equations used in the field.

A. *Energy costs of Reliability*

When considering the overall energy savings of a proposal, the energy consumed in order to provide a reliable computing system must be taken into account. For many systems, this mechanism is a checkpoint/restart. Others may provide reliability through duplicate functional units, or alternative advanced methods, and this must also be taken into account.

Energy saving methods typically result in a reduction in overall system performance, and have a corresponding increase in wall-clock runtime. This increase in runtime will impact the probability that a system failure occurs during the overall runtime of the application.

for P power, α activity factor, C capacitance, V voltage and f frequency,

$$P = \alpha C V f^2 \tag{1}$$

and E energy,

$$E = \int_{start}^{finish} P(t)\, dt \tag{2}$$

for a system with reliability mechanisms, where E_{rel} is the energy required by additional elements or techniques in order to provide reliable operation, E_{rt} is the energy needed for an error free run of the application, the total energy E_{total} is:

$$E_{total} = E_{rt} + E_{rel} \tag{3}$$

and failure probability $p(fail)$ is the sum of the probability of failure during the original runtime $p(fail_{orig\_rt})$ and the probability of failure during the additional runtime imposed by the energy saving method $p(fail_{add\_rt})$:

$$p(fail) = p(fail_{orig\_rt}) + p(fail_{add\_rt}) \tag{4}$$

using the energy required for failure recovery E_{fail_recov} the total energy for a given test is,

$$E_{total} = E_{rt} + E_{rel} + p(fail) \times E_{fail\_recov} \tag{5}$$

where the energy for failure recovery is the energy required for the recovery operations E_{recov_ops} combined with the energy from the lost work and the energy required to redo said lost work E_{lost_work}:

$$E_{fail\_recov} = E_{recov\_ops} + (2 \times E_{lost\_work}) \tag{6}$$

As failures are not typically temporally dependent (over a single application run), the amount of lost work will be distributed evenly over the time between the last recoverable point in the computation. Therefore, this time period can be modeled as the simple 1/2 period of the time between recoverable points in the computation. For a system that uses redundancy techniques like DMR and TMR, the amount of lost work is nil, and the amount of time to recover is very low or zero. However, for techniques such as DMR and TMR, the energy required for reliable operation, E_{rel}, is a multiple of the runtime, E_{rt}. For systems such as traditional checkpoint/restore, the amount of lost work, in terms of time, will on average be equal to 1/2 of the checkpoint period, and the recovery energy, E_{recov_ops}, is equal to equation 7.

$$E_{recov\_ops} = \int_{t=fail}^{t=recov} P(t)\, dt \tag{7}$$

The instantaneous power measurements over the recovery period may differ substantially from those during normal computation, as a given system may be using third level or greater storage systems to retrieve recovery data. If a node-level restart must be performed, but this does not impact the other nodes in the system, there will also be synchronization issues in typical HPC applications, such that the unaffected nodes must wait until the failed node catches up to the next barrier in the computation.

Alternative solutions such as migration via failure prediction can also be used, but the energy consumed by the spare systems required to migrate the running processes away from the failing system must be taken into account as well. Migration also incurs the penalty of creating a runtime synchronicity issue, where the migrated node processes may be significantly behind the other processes running in a massively parallel job, thereby causing stalling of many processes waiting for the migrated processes to catch up.

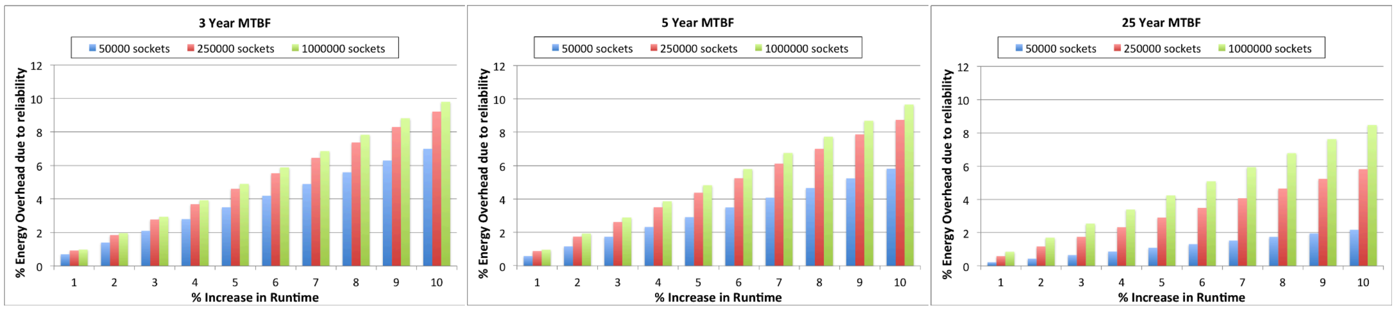
V. Power-Reliability as a Metric

In order to represent the result of these calculations and provide a common point of reference for researchers, we propose a new metric for energy saving techniques, *Energy-Reliability ($E_{ne}R_{el}$)*. This metric illustrates the energy required to run an application in a system where a failure is possible. It adjusts the energy consumed to run an application up by an amount that equals what energy would be consumed when running the application over a long period of time on a real system. It is primarily intended for use with large systems, as individual nodes/sockets typically have MTBFs greater than a realistic application wall clock runtime. $E_{ne}R_{el}$ imposes energy overhead on energy saving methods that increase runtime. It can also be used to determine the possible usefulness of performance enhancing techniques, by quantifying the increased energy consumption vs. the performance enhancement. As it is simply an adjusted value of total energy, $E_{ne}R_{el}$ can be directly used in the most popular and common metrics, energy-delay and energy-delay^2. The authors suggest that such metrics can be denoted as reliability-aware by the use of a subscript r, for example E_rD and E_rD^2 At a very high level, $E_{ne}R_{el}$ can be thought of as:

$$E_{neRel} = E_{rt} + (E_{fail\_recov} * (p(fail) + p(fail_{add\_rt})) \tag{8}$$

It is important to keep in mind that the E_{fail} for an energy saving technique under test will vary from the E_{fail} of the baseline implementation. This is due to the baseline technique having a $p(fail_{add\_rt})$ that is equal to zero. Figures 2a, 2b and 2c show the impact that failures can have on energy consumption given different failure rates and increases in runtime for evenly distributed failures over the runtime. For large node/socket counts the amount of energy that can be lost due to failures in the increased runtime period is not insignificant.

It is important to note that with smaller socket counts that were prevalent in the past, the additional energy required to ensure reliable operation and the amount of lost energy due to failures for a given system was relatively insignificant. However, component counts keep rising and will continue to rise for Exascale computing and beyond. It is well known that reliability will need to improve along with rising component counts in order to make future systems viable. Despite improvements in component reliability, reductions in whole system reliability are still likely to occur. The forecasts in Figure 2, show that



(a) 3 Year MTBF (b) 5 Year MTBF (c) 25 Year MTBF

Fig. 2. Energy overhead due to reliability for varying percentage increases in runtime for 3 year, 5 year and 25 year MTBFs

reliability mechanisms will no longer play a minor role in energy consumption. $E_{ne}R_{el}$ proves useful for illustrating that sometimes spending slightly more energy to finish a job faster, thereby reducing the probability that a failure occurs during the job's runtime, may in some cases actually result in lower energy consumption. The exploration of the effect of reliability on energy consumption is made all the more important by the fact that reliability can be reduced when using energy saving techniques due to thermal cycling [13] and lowered operating voltages result in an increase in soft faults [14].

A. *Determining failure rate*

Modern commodity systems typically have a 3-5 year MTBF per socket, although dependent on the particular technology used, this can vary significantly. By computing the Energy-Reliability of a given system for both 3 year and 5 year MTBFs a graph can be produced that provides reasonable upper and lower bounds for commodity components. It is important to note that the reliability of an Exascale machine (and most large scale capability class systems) will be required to have a higher MTBF of individual components, or a very fast recovery time, to make such a system viable. As was shown in Figure 2c, even with increased reliability, the energy consumption overhead due to reliability will still be significant enough to account for. It is unclear whether reliability will suffer with increased core count per socket, as it does for sockets per node [2]. If reliability does suffer with increased core count, a 1,000,000 socket equivalent system is possible.

Operating processors at reduced frequency results in lower temperatures and reduced rates of failure [3]; therefore DVFS energy saving schemes may result in higher MTBFs. However, it has been shown that for real-time systems, the rate of soft failures caused by radiation increases exponentially as voltage decreases [15]. Energy-saving techniques using DVFS will therefore result in a higher rate of failures due to lowered voltages. The impact of these increased soft failures is difficult to predict. They depend on a variety of factors, one of which is the energy and frequency with which memory and logic gates are hit by energizing radiation. This will vary depending on altitude, geographic location and solar activity. To make accurate predictions as to the increase in soft failure rate as well as the reliability increase due to lowered frequency would be difficult. The methods of determining the effect of voltage scaling on increased failure rates and the increase in reliability due to reduced frequency are outside of the scope of this paper.

B. *Estimating recovery times and energy*

The authors in [16] show that checkpointing and recovery times can be approximated as equal (not assuming incremental checkpointing is occurring). Checkpoint/recovery times are related directly to the I/O times required to write or read the machine's memory to storage. In the case of migration techniques, this is still applicable, but the target is another node's memory, therefore making it network bandwidth dependent. For the purposes of this paper, we will use checkpoint/restore as a baseline for determining energy-reliability. Migration can easily be substituted with appropriate times if the additional spare node energy consumption is taken into account.

When estimating the recovery time for a given system, it is desirable to find an upper bound to the possible energy consumption. Theoretically, the upper bound is the entire memory space of the recovery node(s). This is the most amount of data that could be written during a recovery operation. In practice, the whole memory space rarely needs to be re-written, and should be below 80% of available system memory. Re-writing the entire memory space implies that either all of the contents of memory have changed since the last checkpoint, or that the checkpointing methodology is inefficient.

C. *Best Practices and Actual Practice*

Ideally, all systems and their users would use optimized methods for determining how and when to use reliability techniques. When using duplication techniques like DMR and TMR, much of this complexity is hidden from the user/application developer. However, when using explicit reliability methods like checkpointing or failure prediction and migration, the inputs chosen for the reliability methodology are important to overall efficiency. In reality, application developers and users are not experts in computing reliability and will lack either the skill, time or information required to optimize an application's reliability mechanism efficiency. It is common practice for the users of supercomputers at the U.S. Department of Energy to use a reasonable and familiar time period for checkpointing, most typically one hour. Ideally, the checkpointing interval would be determined as in [17].

D. *Recommended Practice for Energy-Reliability Estimates*

In order to estimate energy-reliability, one must first determine a reasonable number of sockets for a current large scale system by looking at core count and cores per processor. Such information is available at top500.org [18]. This number can then be scaled up and down by half of its value to give a range for current and future systems. One must then calculate the energy-reliability for the systems with appropriate per socket MTBF. Current commodity systems provide a 3-5 year MTBF, but the MTBF for some specialized HPC hardware may be higher.

Using a normalizing the runtime to 10 hours, calculate the additional time that is required due to overhead using equation 9. Using this overhead runtime number, calculate the probability that a failure will happen during the additional runtime with equation 10 and then calculate the energy required for recovery in case of a failure. The probability of a failure must be multiplied by the number of components, this will give the expected number of components that will fail during the additional runtime. The number of failures predicted (hopefully under 1), must then be multiplied by the energy required for recovery. The energy lost due to the failure can be determined by taking the time lost and integrating the average power consumption over the lost time period. The time for the recovery operations themselves may not have a power consumption level that is equal to the average power consumption for the application, it will normally be lower than average calculation power, but this is dependent on the methods being used.

$$t_{add\_rt} = \%_{overhead} \times t_{rt} \tag{9}$$

$$p(fail_{add\_rt}) = 1 - e^{(t_{add\_rt}/MTBF)} \tag{10}$$

E. Energy-Reliability Example

For the November 2012 top500.org list, the system with the largest number of sockets is actually #2 on the list and has 98304 sockets (16 cores per socket). The system happens to be a Blue Gene system, and therefore would have MTBFs of higher than typical values, but for the purposes of this example we will use typical commodity MTBFs. Socket counts for Exascale systems could be as high as 500,000 to 1,000,000 (although unlikely), so a 100,000 socket count is reasonable for estimates on contemporary and near future systems. For simplicity, we round the number of sockets up to 100,000, and therefore we will use 50K, 100K and 150K as our node/socket counts. Runtimes for a given application need to be normalized as well. In this example we have normalized values to use a 10 hour application runtime. Given the phenomenological evidence for checkpointing intervals, we use a 30 minute time period to represent the average ”lost” energy due to a failure. To determine the worst case length of the recovery period, one can determine the time period required to read data to system memory from secondary storage.

If a local storage scheme is adopted, with SSDs running at 500 MB/s, for each of the 98,304 sockets, a recovery time would be 32 seconds. More realistically, having 96 total racks, and 42 U of space per rack, with 1 drive per U of space, and 16,384 GB of memory per rack, the recovery time would be 780 seconds or 13 minutes. This number equates with predictions in existing literature [16] for future systems. Using the established socket count range, recovery and lost work periods with a 5 year MTBF we obtain Figure 3.

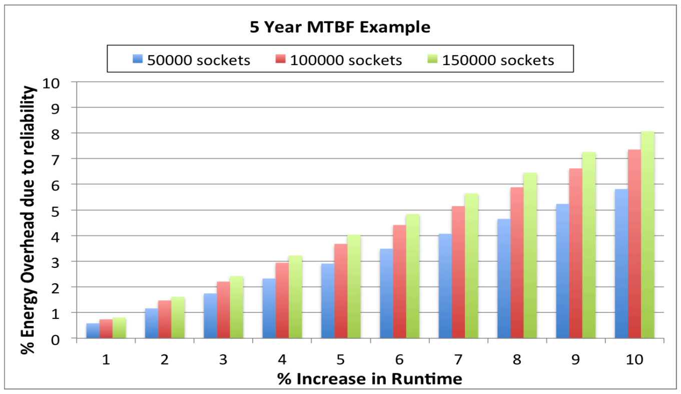


Fig. 3. Energy overhead due to reliability for varying percentage increases in runtime for the example given in section V-E

VI. Model-Based Evaluation of Energy Saving Methods

In order to understand the current state of the art with respect to energy-reliability, this section will examine the publicly available energy saving and overhead numbers for several popular runtime energy saving methods. The original data used to calculate the estimated energy-reliability of the approaches are taken directly from the original publications. References to these publications will be provided along with a summary of the methods in this section. The original published energy savings for each method are adjusted using the model in Section IV to account for reliability, with ideal checkpointing intervals based on the local SSD checkpoint method described in Section V-E.

A. CPUSPEED

CPUSPEED [19] is the default power management system for most Linux distributions. It uses basic power-state governors to dynamically determine when to change power states on a system. CPUSPEED was not designed with HPC in mind and therefore is more aggressive than HPC energy saving techniques in terms of potential energy savings. It provides a baseline for other techniques as it is the default power state adjustment mechanism in Linux.

B. CPU MISER

CPU MISER [3], is a solution that examines execution phases during runtime and makes decisions to lower CPU clock frequencies based on observed phase states. It provides the ability for a user to specify a maximum acceptable slowdown and attempts to adjust the power states such that the total runtime does not fall below the cutoff. As is the case with many runtime energy saving methods, the best energy savings occur for communication bound applications, with limited opportunities to save energy for CPU bound applications.

C. PART

PART [4] was an early entrant into the energy-saving HPC arena. It provided an algorithm for bounding the slowdown of applications while attempting to save energy using the runtime history of an application. It should be noted that the results for PART are only those for the CPU energy consumption, not total system energy. Therefore, the overall energy savings reported for this method will be higher than those for system level energy measurements, as only the CPU energy consumption is reduced using PART.

D. ECOD

ECOD [5] is another performance-bounding and workload predicting algorithm for energy saving. It is more accurate that past methods [3], [4], and provides tighter performance-bounding variance. Unlike PART, this algorithm does not consider the entire workload runtime history when making power state decisions. The energy numbers for ECOD are for a whole system, not CPU-only energy savings like those for PART.

E. NCSU

The method presented in [6] that was developed at North Carolina State University, was not named like many of the other methods surveyed here, so it will be referred to as NCSU. This method concentrates on communication periods and opportunities to reduce CPU power consumption during *blocking* MPI communication. The NCSU method can operate in two modes, one is on-the-fly and the other uses a priori information about the profile of an application. Although the profiling method is more slightly more efficient, the on-the-fly method is used for comparison here, as the NCSU authors did not provide results for the energy used for profiling.

F. Jitter

Jitter [7] provides a method similar to that of NCSU, in that it attempts to exploit slack in MPI programs. Unlike NCSU, it does not exploit communication slack, but that caused by inter-node load imbalance. It requires manual changes to the application source code, and only works for iterative programs.

G. Green Queue

Green Queue [8] is a new (2012) energy savings approach for HPC designed for scalability. It utilizes an intra-node methodology using phase detection through profiling and offline simulation analysis of applications. It uses an SQL database to store profiles on past application runs and simulation analysis. This makes it somewhat different from the other approaches examined as it requires energy consumption to generate the profiling and simulation data. Such overhead is not accounted for in the energy consumption reported for Green Queue.

H. Adagio

Adagio [20] is an integration of multiple methods, including Jitter, that exploits slack in MPI programs for both load imbalance and communication slack. It exploits advanced DVFS techniques to provide improved energy savings by allowing a "task" (a slice of execution between two MPI blocking operations) to be run over multiple frequencies, thereby better approximating the ideal frequency for which that code should be run. It is designed to minimize the performance impact of energy-saving, rather than finding a trade-off between energy consumption and performance loss.

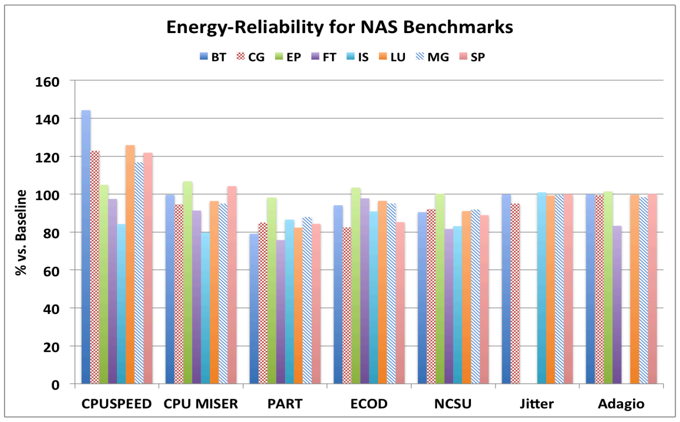


Fig. 4. Comparison of energy saving techniques when accounting for reliability

I. Comparison of Methods

In order to compare the sampling of runtime energy-savings methods, the slowdown and energy saving percentages were collected using the published data for each method. As the implementation of the majority of the methods are not available, the published results are the only results that can be used for a comparison. It should be noted that in this comparison it is assumed that all of the runtime methods scale perfectly, with no additional overhead for large systems. This is very unlikely in practice, so the adjusted energy-reliability of the systems should be considered as a best-case scenario.

Accounting for the extra energy incurred due to failures and subsequent recovery with a 50K socket 3 year MTBF system using ideal checkpointing times, the energy-reliability numbers shown in Figure 4 were determined. It should be noted that the EP and FT benchmark results for Jitter are unavailable, as are the results for the IS benchmark for Adagio. All of the results are shown only for the NAS parallel benchmark suite [21], as they are the only benchmark results common to all of the energy saving methods.

Figure 4 shows the adjusted energy-saving results for six of the seven methods. The results for Green Queue are not included as only the average and best case results are available. Examining the results, we can see that the results for PART, ECOD and NCSU show the best overall energy savings. However, Figure 4 only shows the adjusted energy savings, and does not show the resulting slowdown. Examining the methods using their average performance across the benchmarks, and comparing with the average runtime in Figure 5, one can observe that PART and ECOD sacrifice a higher average runtime than NCSU for approximately the same energy savings. In fairness to PART and ECOD, they are applicable to a larger number of applications than the MPI application only NCSU. The best average runtime slowdown results are those for Adagio. Examining Figure 4 we can see that the energy savings for Adagio are due primarily to excellent energy savings for FT, with little savings for the other benchmarks. Adagio manages good runtime averages by aggressively avoiding slowdown. Due to their runtime overhead impacts, some of the methods see more of an impact when accounting for reliability than others. The percentage increase in overall energy consumption due only to reliability

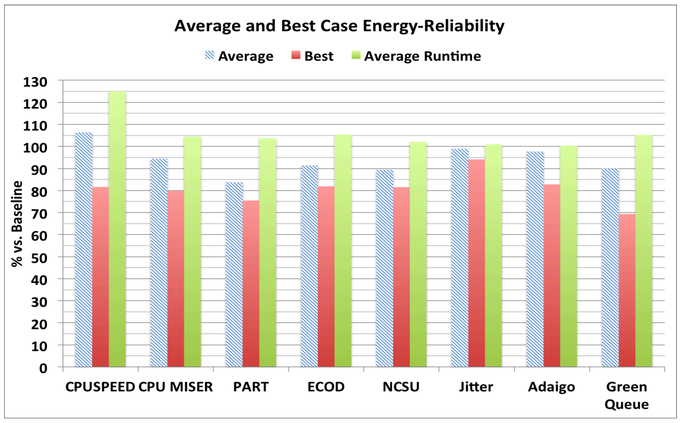


Fig. 5. Average and best case Energy-Reliability with slowdown for all 8 methods

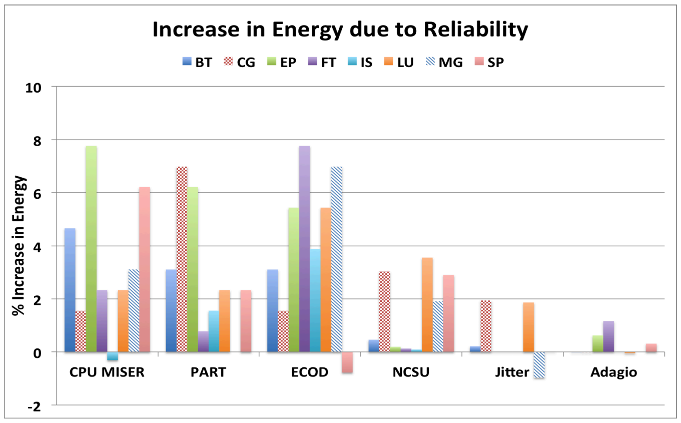


Fig. 6. The percentage increase in energy consumption caused by the probability of failure during the overhead runtime

concerns is presented in Figure 6. CPUSPEED is omitted from this figure for clarity, as it has some increases in energy nearing 40%. By examining Figure 6 it can be observed that the impact of reliability on energy saving methods can be significant enough to warrant consideration. In some cases, the energy consumption results are improved by faster run times. This occurs when slowing down the system results in a favorable reduction in resource contention. It is worth noting that the most recent energy saving methodologies are better at minimizing runtime overhead and therefore have the smallest increases in energy due to reliability.

By adjusting the parameters used in the calculation of energy-reliability, one can observe the conditions under which energy-reliability would not be a significant enough concern in future systems to warrant taking it into account. We will define significant as an impact of more than 1%. With a 25 Year MTBF of individual sockets, the impact of energy-reliability for a million socket system can be reduced to less than 1% for any increase in runtime under 10% with a recovery time of 2 seconds, using ideal checkpointing intervals. For smaller socket counts, we can have longer recovery times and more lost work, for the 500,000 socket case, a 5 second recovery time or less could render energy-reliability as potentially insignificant. For a much smaller system, recovery times can be as long as 45 seconds.

Alternatively, if recovery times stay within the estimates in section V-E and individual component reliability increases, for the million socket system, an MTBF of 1250 years is required. For the 500,000 socket system an MTBF of 625 years is needed, and for the 50,000 socket system, an MTBF of approximately 63 years is needed. Of course, a combination of increased MTBF and smaller recovery times is also a possibility. For example, for a 500K socket system, with an MTBF of 250 years, and recovery time of 45 seconds using ideal checkpointing times to determine lost work; energy-reliability would be rendered insignificant according to the rules outlined previously. These estimates serve to demonstrate that energy-reliability for future systems will likely be a significant enough factor to consider when analyzing any energy-saving methods that cause an increase in runtime.

VII. Short Case Study

To better illustrate how the power-reliability concept can be applied, experiments were conducted using the MAESTRO approach to improving energy-efficiency through concurrency throttling. MAESTRO uses memory concurrency and RAPL-based energy measurements collected by its Resource-Centric Reflection (RCR) daemon to dynamically adjust the number of active worker threads deployed for OpenMP applications running over the Qthreads run time system [22]. When memory bandwidth is saturated, running an active thread on every core is unprofitable, so some threads are throttled by CPU clock modulation and given no more work until and unless reduced memory pressure is detected later in the execution.

A study of single-node OpenMP executions of the LULESH mini-application [23] saw a decrease in energy usage with a very slight increase in execution time [22]. The experiments in [22] were only conducted on a single node. The results shown here extended the experiments to multi-node executions of LULESH using OpenMP+MPI, both the execution time and the energy decreased. In these executions, each 16-core dual-SandyBridge node ran an indepdendent instance of the Qthreads-MAESTRO run time system, but due to the regular structure of LULESH, all nodes throttled at the the same point in the execution. As shown in Figure 7, the energy-reliability model predicts that such a scheme would see an additional 2.9% improvement in energy consumption when large scale system reliability is considered, assuming the method continues to scale well at higher node counts. While further studies are needed, the early results show promise not only for energy savings due to reduction in the number of cores used, but also faster execution times that compound those savings by reducing the time in which the application is exposed to failures.

VIII. Conclusion

This work has detailed a new metric for energy saving methodologies used in large scale systems and shown how this metric can be applied using a straightforward model. It has demonstrated that for large scale systems, additional runtime overhead imposed by energy saving techniques can have significant impact on overall energy savings due to the possibility of reliability events occurring during the runtime overhead period. As power and reliability are both major concerns for current and future large scale capability class

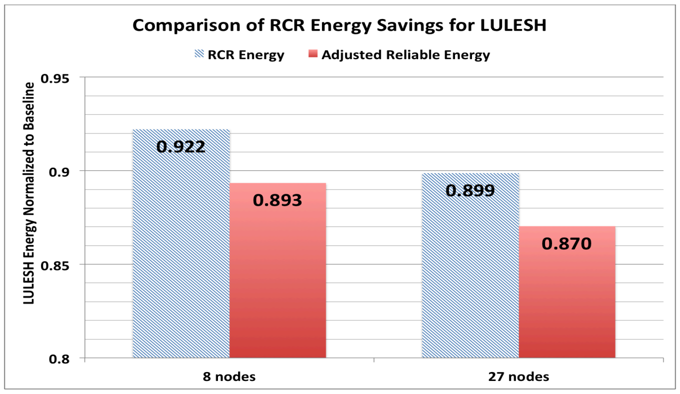


Fig. 7. An example of the adjusted energy savings using energy-reliability for MAESTRO with LULESH

systems, the assessment of proposed energy saving techniques must be done in the context of unreliable systems.

In this paper, a survey of several energy saving techniques for HPC was performed, and each techniques published experimental results were adjusted using the model presented. It was demonstrated that the predicted impact that reliability events will have on energy saving methods should be accounted for, even when perfect scaling of the energy saving techniques is assumed. In addition, the conditions under which energy-reliability would be insignificant were explored. This leads to the conclusion that energy-reliability must be taken into account when exploring runtime energy-saving methods for large-scale capability HPC.

IX. Future Work

We will utilize the concepts presented in this work to analyze the energy efficiency of future runtime energy saving methods. The energy-reliability metric is best applied to very large capability class computing systems. As such we will utilize the metric to aid in studies of energy saving methodologies for Exascale class machines. We are also exploring soft failure rates at lowered voltages and examining methods of collecting and analyzing soft failure rate increases for use in future runtime energy saving research.

Acknowledgment

The authors would like to thank Kevin Pedretti and Kurt Ferreria for their helpful discussions on power and reliability on large scale capability class systems.

References

[1] M. Seager, “Operational machines: ASCI white,” in *7th Workshop on Distributed Supercomputing, Durango, CO*, 2003.

[2] B. Schroeder and G. A. Gibson, “A large-scale study of failures in high-performance computing systems,” *Dependable and Secure Computing, IEEE Transactions on*, vol. 7, no. 4, pp. 337–350, 2010.

[3] 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)*. IEEE, 2007, pp. 18–18.

[4] 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*. IEEE Computer Society, 2005, p. 1.

[5] 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*. IEEE Computer Society, 2009, pp. 68–75.

[6] 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*. IEEE, 2006, pp. 14–14.

[7] 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*. IEEE Computer Society, 2005, p. 33.

[8] 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)*. IEEE, 2012, pp. 260–267.

[9] 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*, 1998, pp. 96–101.

[10] M. Mittal and R. Valentine, “Performance throttling to reduce IC power consumption,” Feb. 17 1998, uS Patent 5,719,800.

[11] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai, “The impact of performance asymmetry in emerging multicore architectures,” in *ACM SIGARCH Computer Architecture News*, vol. 33, no. 2. IEEE Computer Society, 2005, pp. 506–517.

[12] R. E. Grant and A. Afsahi, “Power-performance efficiency of asymmetric multiprocessors for multi-threaded scientific applications,” in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. IEEE, 2006, pp. 8–pp.

[13] J. S. S. T. Association, “Failure mechanisms and models for semiconductor devices,” Tech. Rep., March 2006.

[14] V. Chandra and R. Aitken, “Impact of technology and voltage scaling on the soft error susceptibility in nanoscale CMOS,” in *Defect and Fault Tolerance of VLSI Systems, 2008. DFTVS ’08. IEEE International Symposium on*, 2008, pp. 114–122.

[15] D. Zhu, R. Melhem, and D. Moss, “The effects of energy management on reliability in real-time embedded systems,” in *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, 2004, pp. 35–40.

[16] F. Cappello, H. Casanova, and Y. Robert, “Checkpointing vs. migration for post-petascale supercomputers,” in *Parallel Processing (ICPP), 2010 39th International Conference on*. IEEE, 2010, pp. 168–177.

[17] J. Daly, “A higher order estimate of the optimum checkpoint interval for restart dumps,” *Future Generation Computer Systems*, vol. 22, no. 3, pp. 303–312, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X04002213

[18] H. Meuer, E. Strohmaier, J. Dongarra, and H. Simon, “Top500 supercomputing sites,” 2013. [Online]. Available: http://www.top500.org/

[19] R. Ge, X. Feng, and K. W. Cameron, “Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters,” in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, 2005, p. 34.

[20] 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*. ACM, 2009, pp. 460–469.

[21] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber *et al.*, “The NAS parallel benchmarks summary and preliminary results,” in *Supercomputing, 1991. Supercomputing’91. Proceedings of the 1991 ACM/IEEE Conference on*. IEEE, 1991, pp. 158–165.

[22] A. Porterfield, S. Olivier, S. Bhalachandra, and J. Prins, “Power measurement and concurrency throttling for energy reduction in OpenMP programs,” in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, 2013, pp. 884–891.

[23] I. Karlin, J. Keasler, and R. Neely, “Lulesh 2.0 updates and changes,” Tech. Rep. LLNL-TR-641973, August 2013.