Energy-aware I/O Optimization for Checkpoint and Restart on a NAND Flash Memory System

Takafumi Saito[1,2], Kento Sato[1,3], Hitoshi Sato[1,2] and Satoshi Matsuoka[1,2,4]
[1]Tokyo Institute of Technology
[2]JST/CREST
[3]JSPS Research Fellow
[4]National Institute of Informatics
{t.saito, kent}@matsulab.is.titech.ac.jp, hitoshi.sato@gsic.titech.ac.jp,
matsu@is.titech.ac.jp

ABSTRACT

Both energy efficiency and system reliability are significant concerns towards exa-scale high-performance computing. In such large HPC systems, applications are required to conduct massive I/O operations to local storage devices (e.g. a NAND flash memory) for scalable checkpoint and restart. However, checkpoint/restart can use a large portion of runtime, and consumes enormous energy by non-I/O subsystems, such as CPU and memory. Thus, energy-aware optimization, including I/O operations to storage, is required for checkpoint/restart. In this paper, we present a profile-based I/O optimization technique for NAND flash memory devices based on Markov model for checkpoint/restart. The results based on performance studies show that our profile lookup approach can save 4.1% of energy consumption in an application execution with checkpoint/restart. Especially, our approach improves the energy consumption of write operations by 67.4% and read operations by 40.2% on a PCIe-attached NAND flash memory device.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Fault tolerance; D.4.2 [**Operating Systems**]: Storage Management

General Terms

Reliability

Keywords

Low energy technique; Checkpoint/Restart; NAND flash memory

1. INTRODUCTION

Increased power consumption in modern large-scale supercomputers limits the design and the scale of the systems, and this power limitation is also applied to future extreme scale supercomputing systems. For example, the IESP [10] reports that the upper limit of the power in an exa-scale supercomputer should be less than 20MW, while current power efficient technique requires over 60MW of the system. In such large HPC systems, applications are required to conduct tera- or peta-byte scale massive I/O operations to local storages (e.g. a NAND flash memory) for scalable checkpoint and restart [16]. However, as HPC system size grows, compute nodes perform less computation but consume huge power during checkpoint/restart, which result in undesirable energy consumption. Thus, energy efficiency of I/O operations is important at extreme scale

Recent rapid progress of flash technology introduces a new memory tier in existing computer systems. That is, modern computer systems employ a NAND flash memory, such as SSD or PCIe-attached flash memory, as a padding layer in order to mitigate the performance gap between DRAM and HDD-based secondary storage. Indeed, modern supercomputers such as TSUBAME2.0 and Gordon employ flash-based storages, and these NAND flash technologies may also improve checkpoint and restart I/O performance in terms of throughput and latency for various HPC applications. These devices can be applied to storage volumes for checkpoint and restart I/Os instead of a traditional parallel file system (PFS); however, the validity of energy efficiency of checkpoint and restart I/O to these devices is not clear.

To address the above problem, we investigate whether energy on checkpoint and restart I/O can be reduced by applying DVFS(dynamic voltage and frequency scaling) and controlling I/O processes under the specific system fault rate. In order to estimate checkpoint and restart time, and to optimize energy consumption, we model an application runtime state using Markov model based on the existing model [21]. The results based on performance studies show that our profile lookup approach can save 4.1% of energy consumption during the application execution with checkpoint/restart. Especially, in Fusion-io's ioDrive [1], our approach can improve the energy consumption of write operations by 67.4% and read operations by 40.2%.

2. BACKGROUND

2.1 Checkpoint and Restart

The overall failure rate of HPC systems increases with the size of the HPC system. Checkpoint/restart is a widely-

FTXS'13, June 18, 2013, New York, NY, USA.

used fault tolerant technique used by HPC applications that typically run continuously for hours or days at a time. A checkpoint is a snapshot of an application state, which is usually written to a reliable persistent storage volume such as a PFS. On a failure, checkpoints are used to restore the application states so that the application can restart the execution form the latest checkpoint. However, when writing checkpoint files to PFS in large-scale systems, the I/O performance of PFS can be a bottleneck, which causes huge overheads to application runtime [7, 18].

Diskless checkpoint [19, 11] is one of the solutions to minimize the overheads. The checkpoint technique writes application state to distributed node-local storage devices without relying on PFS, so the approach can eliminate overheads by traditional checkpoint techniques. In fact, since most of the failures are caused by one or a few nodes at a time [20, 16], the diskless checkpoint approach can recover from most of failures while minimizing checkpoint overheads.

However, even with diskless checkpointing, checkpoint time increases as system size and failure rate increase in future extreme scale systems. Additionally, during checkpoint/restart, computing nodes perform less computation but consume huge power. Thus, the energy consumption increases even during I/O operations in future extreme scale systems, energy-aware I/O optimization is important in checkpoint/restart.

2.2 NAND flash memory

Existing studies [16, 11] minimize the impact of checkpoint using NAND flash memory devices, such as solid state disk (SSD), PCIe-attached flash storage. A NAND flash memory device is one of the non-volatile memory devices in which data is stored in an array of memory *blocks*. A block contains multiple *pages*, each of which is the smallest size of read and write operations. Although a NAND flash memory device exhibits random read operations as fast as sequential read operations by eliminating mechanical head movement like traditional HDDs, random write operations are not as fast as sequential ones. This performance difference is due to the mechanical feature of a NAND flash memory device. In a NAND flash memory device, overwrite operations are prohibited, and pages are written sequentially within a block. Thus, when writing a new page, pages that are not overwritten need to be moved out from the block, and those pages are written to another available block sequentially with the new page. Therefore, in order to exploit the peak I/O performance of these NAND flash memory devices, an application needs to write the checkpoint sequentially.

When we apply energy-aware I/O optimization techniques to NAND flash memory devices, especially PCIe-attached flash memory devices, we need to consider the unique features. For example, in Fusion-io ioDrive [1], which is a widely used PCIe-attached flash memory devices, *grooming* and *wear leveling* techniques are used in order to optimize write operation. Grooming is a garbage collector that pre-erases unused blocks in background so that future overwrite operations can omit erase operations to the target blocks. Wear leveling is a technique to extend the lifetime of a device. Data can be written to addresses on a flash device with a finite number of times. The limited number of write operations is large enough (typically 10,000 or 100,000 [15, 12]), however if a system writes to the same portion of the memory device over and over again, only that portion wears out. To avoid the unbalanced wear-out, flash devices use the wear leveling technique.

In recent PCIe-attached flash memory devices, grooming and wear leveling operations rely on CPU cores. If we simply apply DVFS to the PCIe-attached flash memory devices to decrease CPU frequency during I/O operations, we can save the power consumption. However, such a DVFS technique may cause prolonged I/O time and result in high energy consumption. In addition, if we write data to a PCIe-attached memory device with multiple processes concurrently, we can utilize the bandwidth of the device; however, such operations can be random write operations from the device's perspective, whose situation also causes prolonged I/O time and consumes undesirable energy. Thus, optimization of both CPU frequencies and numbers of concurrent I/O processes is important to save energy consumption during checkpoint/restart at future extreme scale.

3. RELATED WORK

As system size and failure rate in HPC systems increase, checkpoint/restart time and checkpoint frequency also increase. Additionally, computing nodes perform less computation but consume huge power, an energy-aware I/O optimization is significant. Existing studies reduce I/O energy consumption by utilizing local storage devices [14, 17]. PFSs usually consume more energy than local storage devices, and one of the promising approaches is to use levels of storage hierarchically according to the performance and the energy consumption. Manzanres et al. [14] have proposed an algorithm that prefetches popular data from a PFS into a local storage device to reduce accesses to the PFS based on an energy-saving prediction model. Nijim et al. [17] have proposed an energy efficient hybrid parallel disk system, whose technique integrates flash memory devices as a cache to achieve energy efficiency in a PFS. A flash memory device itself is an energy efficient device, exploiting flash memory devices is important for building energy efficient storage systems.

Another promising approach is to use DVFS techniques during I/O operations. Ge et al. [9] integrates DVFS control at the parallel I/O middleware layer to reduce energy consumption of PFSs. The middleware analyses a I/O pattern, and determines the optimal CPU frequency, and applies DVFS in order to minimize energy consumption based on a I/O profile tool [8]. This work focuses on a DVFS approach. However, as described, when we apply an energy-aware I/O optimization to a NAND flash memory, the number of concurrent I/O processes must be considered as well as CPU frequency.

Optimizing parallel I/O performance also contributes to improve I/O energy efficiency. Hasan et al. [3] have achieved high I/O throughput by using RDMA (remote direct memory access) with additional nodes. Nawab et al. [4] have proposed a technique that asynchronously transfers multiple striped data streams to increase I/O performance in Grid environments. Asynchronous I/O can utilize CPU not being used during checkpoint/restart, which results in energy efficiency because I/O operations can be hidden.

Our focus is to minimize energy consumption during checkpoint/restart. However, a DVFS technique can be also applied to a general application execution. Wang et al. [22] decrease energy consumption of task executions by increasing task execution time within affordable limits using DVFS.

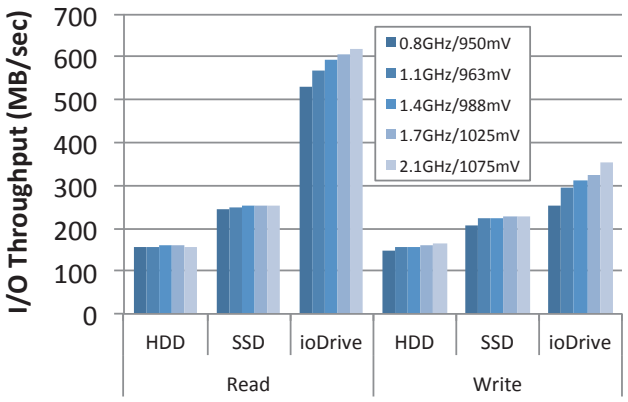

Figure 1: I/O throughput with different CPU frequencies and devices. (1 process)

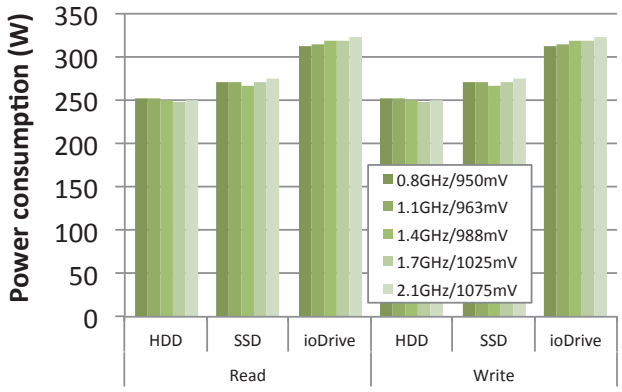

Figure 2: Power consumption with different CPU frequencies and devices (1 process)

Huang et al. [13] proposed energy efficient scheduling algorithm that reduces energy consumption by slacking non-critical jobs while keeping performance-based service level agreement. Actually, our approach is independent of the above techniques. So, by combining those DVFS algorithms with our I/O profile approach, we can improve energy efficiency of both application runtime and checkpoint/restart.

4. POWER/PERFORMANCE EVALUATION OF CHECKPOINT/RESTART

Table 1: Node specification

CPU	AMD Opteron Processor 6172 (12 cores) × 4 sockets
Memory	DDR3-1333 SDRAM DIMM (128GB)
HDD	Fujitsu MHZ2500B (rpm:4200, seek:12ms)
SSD	Intel SSD 320 Series 600GB, SSDSA2CW600G3K5 (Sequential read/write: 270/220MB/s)
PCIe-attached flash memory	Fusion-io ioDrive MLC 320GB (Read/Write bandwidth: 735/510MB/s)

As a first step for energy-aware checkpoint/restart I/O optimization, we run a micro benchmark to investigate how much power consumption and I/O performance may change by (1) CPU frequencies, (2) numbers of concurrent I/O processes, (3) I/O operation types, i.e., read or write, (4) target storage devices, i.e., HDD, SSD, and PCI-attached flash devices such as ioDrive. Specification of the machine we use is shown in Table 1. Specifically, we use Fusion-io's ioDrive, a promising I/O device for HPC applications [6], in this micro benchmark. The micro benchmark divides a 2GB file into the number of specified processes, and each of the processes sequentially reads or writes the corresponding file chunk. The system optimizes energy consumption of checkpoint/restart, so we target only sequential read and write operations with Mbytes and Gbytes order of files.

Figure 1 and Figure 2 show the results of read and write throughput under the given CPU frequency and power voltage. The number of I/O processes is set to one. We use *cpufreq* [5] for scaling CPU frequency and power voltage. *cpufreq* is a system software which can adjust CPU frequency and power voltage on the fly. We measure the power consumption of the entire machine by using OMRON RC 3008 [2], which samples power consumption and can be remotely monitored every second. Here, we see that the I/O performance of HDD and SDD is almost constant for any CPU frequencies, while ioDrive exhibits the performance degradation with decreasing CPU frequencies. Especially, if we change the CPU frequency from 2.1GHz to 0.8GHz, the degradation of the write throughput significantly decline; 29% decline compared with the peak CPU frequency (2.1Gz). In respect to power consumption, we see that HDD and SSD exhibit almost constant results for any CPU frequencies; however, power consumption of ioDrive increase when we increase CPU frequency.

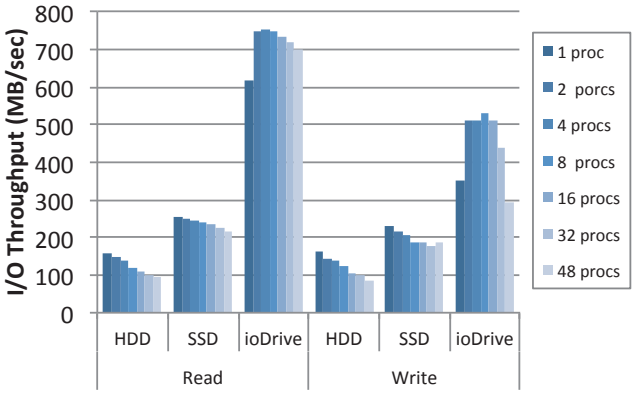

Figure 3: I/O throughput with different number of processes and devices

Figure 3 and Figure 4 show the results of I/O throughput and power consumption under the given parameters; the CPU frequency is set to 2.1 GHz, and the power voltage is set to 1075mV respectively. Here we observe that HDD and SSD exhibit nearly peak performance in both read and write operations with a single I/O process. On the other hand, ioDrive has the optimal number of concurrent I/O processes to exploit device's I/O bandwidth. Especially, when we in-

Figure 4: Power consumption with different number of processes and devices

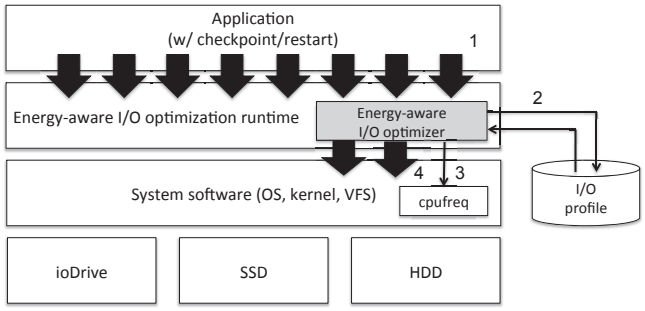

Figure 5: Intended system overview

crease the number of the processes over 16, the throughputs of both read and write operations starts declining. In respect to power consumption, we see that HDD exhibits constant results for any process counts; however, the results in SSD and ioDrive become high when we increase the number of processes. Specifically, we observe that the power consumption of ioDrive jumps sharply.

As described above, I/O performance and power consumption depend on CPU frequencies, the number of concurrent I/O processes, I/O operation types, and target storage devices. Specifically, even the same NAND flash memory devices, i.e., SSD and ioDrive, exhibit significantly different I/O performance and power consumption trend. For example, if an application writes its checkpoint to ioDrive at high parallelism as when doing computation (e.g. 48 processes), the write throughput becomes the worst as in Figure 3 and consumes the most power as in Figure 4, which results in high energy consumption while there is no such impact on SSD. When we apply energy-aware optimization techniques to NAND flash memory devices, the trend has to be considered as the number of CPU cores in a machine increases in future extreme scale systems.

5. OVERVIEW OF ENERGY-AWARE I/O OPTIMIZATION SYSTEM

As mentioned in Section 2.2, performance and power consumption of checkpoint I/O vary according to CPU frequencies and numbers of I/O processes. In order to optimize these parameters and to minimize energy consumption, we design an energy-aware I/O optimization runtime system.

Figure 5 shows the overview of the intended architecture. In this work, we employ a profile lookup approach. That is, when an application starts checkpointing/restart, i.e., write/read operations, the runtime system retrieves *I/O types* (write/read) and *paths* to distinguish devices to access (Step 1). The runtime system looks up the *I/O profile* to set the target CPU frequency and the number of concurrent write/read processes (Step 2), and applies DVFS control via *cpufreq* interface provided by Linux kernel (Step 3). Finally, the runtime system increases or decreases I/O parallelization to adjust the target concurrent number of processes (Step 4).

The profile takes a CPU frequency and the number of processes as the input, gives us checkpoint/restart throughput, and estimated power consumption (W_C/W_R) as the output so that we can compute checkpoint/restart time (T_C/T_R) given checkpoint/restart data size retrieved by the runtime system. The optimal CPU frequency and the number of I/O processes for energy efficiency are determined by our checkpoint/restart model and optimization technique (see Section 6).

We create the I/O profile once before executions of an application, and the I/O profile is constructed by our preliminary evaluation in Section 4. In general, I/O intensive applications can conduct a variety of small and large write/read operations during its overall runtime, the throughput and the energy consumption can vary depending on write/read size. But we focus on optimizing energy consumption of checkpoint/restart to local storages, whose checkpoint/restart data size is generally large enough (order of Mbytes, Gbytes), and the access pattern is sequential, so throughput and energy consumption of I/O do not significantly change every checkpoint/restart. Thus, creating the I/O profile once before executions of an application is enough to obtain accurate W_C/W_R and T_C/T_R.

In this paper, we focus on the investigation of how much the profile lookup approach can improve the energy efficiency of checkpoint I/O for NAND flash memory devices.

6. ENERGY CONSUMPTION MODELING AND OPTIMIZATION

When we execute an application while taking checkpoints, the application runs for a fixed time period (*application runtime state* denoted by *state A*) and writes a checkpoint (*checkpoint state* denoted by *state C*). If a failure happens in state A or state C, the application tries to restore the most recent checkpoint (*restart state* denoted by *state R*), and to restart the execution for state A. If no failure happens in state A and state C, the application running for state A can transition to the next state A. Thus, when an application takes a checkpoint and restores the status from the checkpoint in the case of a failure, the *actual runtime*, including the times for application execution, checkpoint, and restore, become longer than the *ideal runtime* where the application runs with no failures and no checkpoint/restart operations.

In order to estimate and minimize energy consumption of an application execution with checkpoint/restart, we model the actual runtime by using Markov model based on the existing Vaidya's model [21]. Actually, most of the failures are caused by one or a few nodes, which can be recovered from

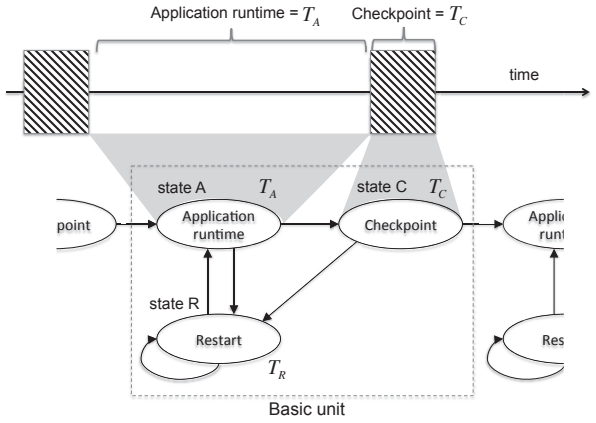

Figure 6: Basic unit of checkpoint a markov model

a local storage checkpoint [20]. We model a local storage checkpoint case as a target checkpoint level.

Figure 6 shows a basic unit of a checkpoint interval. An application can transition across the basic units in sequence, while changing the state within a basic unit. T_A denotes an effective application runtime in which the application can proceed the meaningful work without any extra operations such as checkpoint/restart, i.e., between the end of the latest checkpoint and the beginning of the next checkpoint; T_C and T_R denote checkpoint and restart times. As described, if a failure happens, the running application must roll-back to the latest checkpoint, and compute the same instructions which may have been done before the failure. Thus, the *actual runtime* of the application execution with checkpoint and restart, i.e., the mean sojourn times of each state ($T_{\bar{A}}$, $T_{\bar{C}}$ and $T_{\bar{R}}$), may become larger than the *ideal runtime*, T_A. Given a system failure rate λ, the Vaidya's model gives each of the mean sojourn times as follows:

$$\begin{aligned} T_{\bar{A}} &= \lambda^{-1} e^{\lambda(T_C+T_R)}(e^{\lambda T_A}-1) \\ T_{\bar{C}} &= \lambda^{-1}(e^{\lambda T_C}-1) \\ T_{\bar{R}} &= \lambda^{-1}(e^{\lambda T_C}-1)(e^{\lambda T_R}-1) \end{aligned}$$

Since we use the Vaidya's model, we make the same assumptions as in the model. This formula means that an application costs $T_{\bar{A}} + T_{\bar{C}} + T_{\bar{R}}$ of the time to compute for effective runtime; T_A. Here, we call T_A as *effective runtime* to differentiate from these mean sojourn times. Actually, given checkpoint time (T_C) and failure rate (λ), an optimal checkpoint interval can be obtained as $T_A = \sqrt{2 \times T_C/\lambda}$ by the Vaiday's model. Thus, $T_{\bar{A}}$, $T_{\bar{C}}$ and $T_{\bar{R}}$ are determined by T_C, T_R and λ. Based on the model, an expected total energy consumption in a basic unit, J, can be obtained as $J = T_{\bar{A}} \cdot W_A + T_{\bar{C}} \cdot W_C + T_{\bar{R}} \cdot W_R$, where W_A, W_C and W_R are power consumptions of the application runtime, checkpoint time and restart time in a basic unit, respectively. Here, our focus in this paper is to optimize the energy consumption by applying DVFS and by controlling I/O processes for NAND flash memory devices. Thus, the optimized total energy consumption within a checkpoint interval, J_{opt}, is obtained as:

$$J_{opt} = minimize\{J\}$$

Because an application transitions across the same basic units, we can optimize the total energy consumption by minimizing energy consumption of a single basic unit; J. As described in Section 2.2, checkpoint/restart time (T_C/T_R) and the power consumption during the checkpoint/restart (W_C/W_R) vary according to CPU frequencies and numbers of concurrent I/O processes. Based on the I/O profile, we can find the optimal CPU frequency and the number of I/O processes for checkpoint/restart to minimize J.

7. EVALUATION

7.1 Evaluation setting

We investigate how much our profile lookup approach can improve energy consumption by using a checkpoint/restart technique for NAND flash memory devices. We use 64GB of a file per a compute node for checkpoint/restart and 471.1 watts of an application's power consumption in a single compute node, whose parameters are based on SP (Class C) of NAS Parallel Benchmarks running with 48 processes on a machine shown in Table 1. We set the system failure rate to $\lambda = 1.89474 \times 10^{-5}$ (MTBF≈14hours), which is the same parameter as the average failure rate of TSUBAME2.0 in the year and a half from November 1st 2010 to April 6th 2012. We compare our I/O profile lookup approach to other DVFS strategies provided by a *cpufreq* governor: performance, powersave and ondemand. Details of each power save strategy are show in Table 2.

Table 2: Compared cpufreq governor

Performance	Set CPU frequency to maximum supported frequency regardless of CPU usage by cpufreq driver
Powersave	Set CPU frequency to the lowest supported frequency regardless of CPU usage by cpufreq driver
Ondemand	Adjust CPU frequency according to CPU usage by cpufreq driver
Profile lookup	Adjust CPU frequency and the number of concurrent I/O processes according to our energy-aware profile

7.2 Energy efficiency comparison

In order to evaluate the energy efficiency of an application execution with checkpoint/restart I/O operations, we introduce an energy efficiency index for effective application execution named *energy consumption per unit time for effective application execution (EPE)*. EPE quantifies how much energy is consumed for an effective application execution, i.e., *effective runtime*, T_A, whose expression is described as:

$$EPE = \frac{T_{\bar{A}} \cdot W_A + T_{\bar{C}} \cdot W_C + T_{\bar{R}} \cdot W_R}{T_A}$$

Figure 7 shows the results of EPE in different DVFS strategies and devices. Here we see that our profile lookup technique can save 1.5% of energy consumption on SSD, 4.7% on ioDrive by only considering energy efficiency of I/O operations, because our technique can optimize both CPU frequencies and numbers of I/O processes, and contribute I/O performance improvement. Table 3 shows the results of CPU frequencies and numbers of I/O processes selected by our

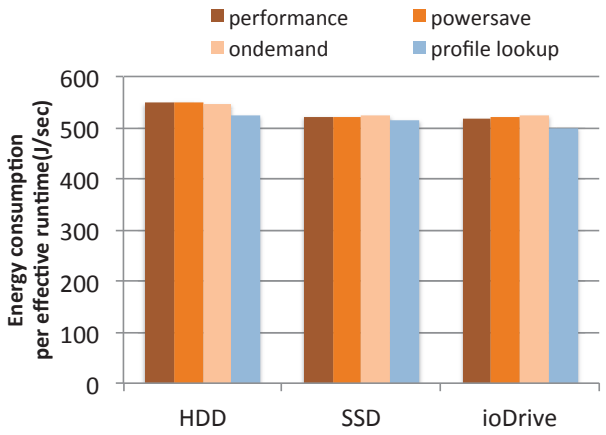

Figure 7: Energy consumption per unit time for effective application execution (EPE) in different DVFS strategies and devices with 48 processes

Table 3: Selected CPU frequency & the number of I/O processes by Profile lookup approach

	Read			Write		
	HDD	SSD	ioDrive	HDD	SSD	ioDrive
CPU frequency	1.7	1.4	2.1	2.1	1.7	1.4
# of processes	1	1	2	1	1	2

profile lookup approach. We found that we can optimize energy consumption for checkpoint/restart I/O by using one or two I/O processes. The performance improvement in an application execution is limited in Figure 7; however, applications running on future extreme scale systems, where failure rate of the system and checkpoint/restart cost may increase, require us to use more checkpoint/restart time than that in current systems. Thus, we believe that our profile lookup approach is expected to be more effective in extreme scale systems. Figure 8 shows energy consumption per MB for write/read operations. We found that energy consumption of write operations on SSD and ioDrive are the same extent. Fusion-IO ioDrive is known as an energy efficient I/O device; however, without energy-aware I/O optimization, we can not exploit peak I/O performance of ioDrive. This result is also applicable to NAND flash memory devices that employ process-based grooming and wear leveling techniques. In especially ioDrive, a checkpoint operation consumes much energy without scaling down the number of write processes on checkpoints. This is because a write operation with 48 processes consume highest power consumption, and exhibit lowest throughput in any number of processes as in Figure 3, 4, which results in high energy consumption. Our profile lookup approach can improve the energy consumption in any types of I/O operations and different devices. Especially, our profile lookup approach improves the energy consumption of write operations by 67.4% and read operations by 40.2% in ioDrive, which is expected to contribute energy efficiency at extreme scale.

8. CONCLUSION

To minimize energy consumption on checkpoint/restart I/O, we proposed a profile lookup approach and investigated

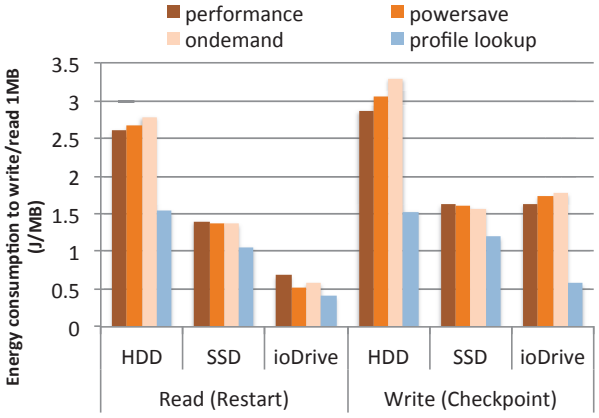

Figure 8: energy consumption per MB for write/read operations with 48 processes

how much energy can be reduced by applying DVFS and by controlling I/O processes. In order to estimate checkpoint and restart time and to optimize energy consumption of an application execution with checkpoint/restart operations, we model application's states using Markov model. The results based on performance studies show that our profile lookup approach can save 4.1% of energy consumption during the application execution with checkpoint/restart. Especially, in ioDrive, our approach can improve the energy consumption of write operations by 67.4% and read operations by 40.2%. As a part of future work, we will extend the profile lookup approach to be able to optimize energy consumption of general data-intensive applications by supporting other I/O types such as random I/Os and stride I/Os.

9. ACKNOWLEDGMENTS

This work was supported by Grant-in-Aid for Scientific Research (S) 23220003, the Japan Science and Technology Agency (JST), and the Core Research of Evolutionary Science and Technology (CREST) research project.

10. REFERENCES

[1] Fusion-io. http://www.fusionio.com/.
[2] OMRON RC3008.
[3] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng. DataStager: Scalable Data Staging Services for Petascale Applications. In *Proceedings of the 18th ACM international symposium on High performance distributed computing*, HPDC '09, pages 39–48, New York, NY, USA, 2009. ACM.
[4] N. Ali and M. Lauria. Improving the Performance of Remote I/O Using Asynchronous Primitives. pages 218–228.
[5] D. Brodowski and N. Golde. "Linux CPUFreq - CPUFreq governors," Linux Kernel. http://www.mjmwired.net/kernel/Documentation/cpufreq/governors.txt.
[6] A. M. Caulfield, J. Coburn, T. Mollov, A. De, A. Akel, J. He, A. Jagatheesan, R. K. Gupta, A. Snavely, and S. Swanson. Understanding the impact of emerging non-volatile memories on

high-performance, io-intensive computing. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–11, Washington, DC, USA, 2010. IEEE Computer Society.

[7] E. N. E. et al. System Resilience at Extreme Scale, Technical report. Technical report, 2008.

[8] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *Parallel and Distributed Systems, IEEE Transactions on*, 21(5):658 –671, may 2010.

[9] R. Ge, X. Feng, and X.-H. Sun. Sera-io: Integrating energy consciousness into parallel i/o middleware. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, CCGRID '12, pages 204–211, Washington, DC, USA, 2012. IEEE Computer Society.

[10] A. Geist and S. Dosanjh. Iesp exascale challenge: Co-design of architectures and algorithms. *Int. J. High Perform. Comput. Appl.*, 23(4):401–402, Nov. 2009.

[11] L. A. Gomez, N. Maruyama, F. Cappello, and S. Matsuoka. Distributed Diskless Checkpoint for Large Scale Systems. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 63–72. IEEE, May 2010.

[12] L. Grupp, A. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. Siegel, and J. Wolf. Characterizing flash memory: Anomalies, observations, and applications. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 24 –33, dec. 2009.

[13] Q. Huang, S. Su, J. Li, P. Xu, K. Shuang, and X. Huang. Enhanced energy-efficient scheduling for parallel applications in cloud. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 781 –786, may 2012.

[14] A. Manzanres, X. Ruan, S. Yin, M. Nijim, W. Luo, and X. Qin. Energy-aware prefetching for parallel disk systems: Algorithms, models, and evaluation. In *Network Computing and Applications, 2009. NCA 2009. Eighth IEEE International Symposium on*, pages 90 –97, july 2009.

[15] N. Mielke, T. Marquart, N. Wu, J. Kessenich, H. Belgal, E. Schares, F. Trivedi, E. Goodness, and L. Nevill. Bit error rate in nand flash memories. In *Reliability Physics Symposium, 2008. IRPS 2008. IEEE International*, pages 9 –19, 27 2008-may 1 2008.

[16] A. Moody, G. Bronevetsky, K. Mohror, and B. R. de Supinski. Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–11, Washington, DC, USA, Nov. 2010. IEEE Computer Society.

[17] M. Nijim, A. Manzanares, X. Ruan, and X. Qin. Hybud: An energy-efficient architecture for hybrid parallel disk systems. *Computer Communications and Networks, International Conference on*, 0:1–6, 2009.

[18] 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 *Proceedings of the 24th IEEE Conference on Mass Storage Systems and Technologies*, MSST '07, pages 30–46, Washington, DC, USA, 2007. IEEE Computer Society.

[19] J. S. Plank, K. Li, and M. A. Puening. Diskless Checkpointing. *IEEE Trans. Parallel Distrib. Syst.*, 9(10):972–986, Oct. 1998.

[20] K. Sato, N. Maruyama, K. Mohror, A. Moody, T. Gamblin, B. R. de Supinski, and S. Matsuoka. Design and modeling of a non-blocking checkpointing system. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[21] N. H. Vaidya. On Checkpoint Latency. Technical report, College Station, TX, USA, 1995.

[22] L. Wang, G. von Laszewski, J. Dayal, and F. Wang. Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with dvfs. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 368–377, Washington, DC, USA, 2010. IEEE Computer Society.