Application-Based Coarse-Grained Incremental Checkpointing Based on Non-volatile Memory

Zhan Shi(✉), Kai Lu, Xiaoping Wang, Wenzhe Zhang, and Yiqi Wang

College of Computer, National University of Defense Technology, Changsha, People's Republic of China
alec_1994@126.com, lukainudt@163.com, {xiaopingwang,zhangwenzhe}@nudt.edu.cn, 459046923@qq.com

Abstract. The Mean Time to Failure continues to decrease as the scaling of computing systems. To maintain the reliability of computing systems, checkpoint has to be taken more frequently. Incremental checkpointing is a well-researched technique that makes frequent checkpointing possible. Fine-grained incremental checkpointing minimizes checkpoint size but suffers from significant monitoring overhead. We observe the memory access at page granularity and find that the size of contiguous memory regions visited by applications tends to be proportional to size of corresponding memory allocation. In this paper, we propose the Application-Based Coarse-Grained Incremental Checkpointing (ACCK) that leverages the priori information of the memory allocation to release the memory monitoring granularity in an incremental and appropriate way. This provides better opportunities for balancing the tradeoff between monitoring and copying overhead. ACCK is also assisted by hugepage to alleviate the TLB overhead. Our experiment shows that ACCK presents 2.56x performance improvement over the baseline mechanism.

1 Introduction

As more of high performance computing systems are being deployed in practice, fault-tolerance and self-healing are becoming increasingly important. Large-scale machines are often used by the scientific community to solve computationally intensive applications typically running for days or even months. Fault-tolerance is also a must-have characteristic of large-scale software system used in safety-critical and business-critical domains. However, large systems exponentially increase the total number of failure points and suffer [1, 2]. For instance, if mean-time-to-failure (MTTF) of an individual node is as long as ten years, the MTTF of 64,000 node system is expected to decrease to the order of tens of minutes [5, 12]. Therefore, it is critical to tolerate hardware and software failures with low impact on application performance and avoid re-computation.

Checkpoint-restart is an effective mechanism for preventing applications from restarting from scratch when unexpected system failure or scheduled maintenance occurs [3–5]. A checkpoint is a snapshot of application state in non-volatile devices used for restarting execution after failure. However, the scaling of computing system puts additional pressure on checkpoint mechanism. On the one hand, checkpoint has to be

G.R. Gao et al. (Eds.): NPC 2016, LNCS 9966, pp. 45–57, 2016.
DOI: 10.1007/978-3-319-47099-3_4

taken more frequently [6, 7] relative to the failure rate of the system which, in turn, directly impacts the application execution time and non-volatile storage requirements. On the other hand, the overhead of global checkpoint increases due to the large checkpoint size.

With the emerging non-volatile memory technologies, fast in-memory checkpoint helps alleviate the data transfer bandwidth problem, which is recognized as a major bottleneck of checkpointing. Moreover, in-memory approach allows frequent checkpointing to meet the requirements of large-scale systems. The presence of non-volatile memory provides opportunities for incremental checkpointing technique. Since the in-memory checkpointing makes it possible to take checkpoints every few seconds, it reduces the overhead of incremental checkpointing. As the checkpoint interval decreases, a small number of memory blocks are modified, hence, the average size of an incremental checkpoint decreases. Recent work [3–5] has demonstrated that the incremental checkpointing shows its advantage over full-size checkpointing when the checkpoint interval is within one minute.

To employ an incremental checkpointing technique, dirty page management is required for every page in the DRAM. Page-based incremental checkpointing techniques leverage the hardware page protection to track dirty pages and back up the dirty pages at the end of checkpoint interval [8]. Generally, the overhead of incremental checkpointing is made up of two parts, the monitoring overhead and the copying overhead. These two parts are a pair of trade-off factors and the key to balance the tradeoff is the granularity of the monitor. Finer-grained monitoring identifies the modified data in finer granularity with higher monitoring overhead, but is able to copy less unmodified data. With coarser-grained monitoring, the monitoring overhead is reduced but more unmodified data needs to be copied. Our experiment reveals that page-based incremental checkpointing technique suffers more from significant monitoring overhead. It indicates that a coarser-grained implementation will better balances the tradeoff and boosts the overall performance. However, blindly releasing the granularity does benefit to the monitoring overhead anyway, but results in extremely large checkpoint size which slow both the checkpoint and restart processes.

We observe the memory access characteristics at page granularity and find that the size of contiguous memory regions visited by applications tends to be proportional to the size of corresponding memory allocation. It indicates that the memory allocation information can be leveraged to appropriately release the monitoring granularity. Based on this observation, we propose Application-Based Coarse-Grained Incremental Checkpointing (ACCK) mechanism that strikes the balance between the monitoring overhead and copying overhead in incremental checkpointing. Based on the relationship between contiguous memory regions visited by applications and the corresponding memory allocation, ACCK mechanism releases the monitoring granularity by not monitoring the memory regions that the applications tend to visit. Since page-based incremental checkpointing works with normal-size pages to enable fine-grained monitoring, it inevitably increases the TLB overhead. We implement merge and split operation in Linux kernel which merges normal-size pages (4 Kbyte) to huge-size page (2 Mbyte) and splits huge-size page back to normal-size pages. ACCK merges normal-size pages when it decides to keep them unmonitored and splits huge-size page at checkpoint time to maintain fine-grained monitoring. This effectively alleviates the TLB overhead during the runtime.

Compared to traditional page-grained incremental checkpointing, ACCK mechanism largely reduces the monitoring overhead with bearable increase in copying overhead. The experimental results show that the performance of memory access monitoring, which accounts for around 70 % of the checkpointing time, improves 9x on average. As a result, ACCK presents a 2.5x performance improvement over the baseline page-grained based incremental checkpointing mechanism with the copying overhead increases by 7 % and the TLB overhead alleviated.

The remainder of this paper is organized as follows: Sect. 2 introduces the background and our motivation. We describe the design and implementation of ACCK in detail in Sect. 3. Section 4 gives the experiments results. Related work is discussed in Sect. 5 and we conclude in Sect. 6.

2 Background and Motivation

2.1 Non-volatile Memory

Non-volatile memory (NVM) is maturing fast in recent years. NVM features fast access, non-volatile, large capacity, and byte-addressable. Currently, phase change memory (PCM) is the most developed non-volatile memory technology [8–10]. The read speed of PCM is comparable with DRAM, while the write speed is much slower (typically 10x slower). There are many studies at architecture level focusing on mitigating the write latency [11, 12]. In this paper, we emulate the slow write by adding latency to test its performance on supporting checkpoint.

HDD data transfer bandwidth has long been a main concern for checkpointing technologies [11]. NVM allows in-memory copy to accelerate checkpointing. With the NVM support, the age-old checkpoint/restart mechanism seems more attractive due to its simplicity and low cost. Recent work [11] has shown that PCM-supported in-memory checkpointing is 50 times faster than the conventional in-HDD checkpointing. We assume the non-volatile memory can be directly accessed by CPUs via load and store instructions. It shares the same physical address with DRAM.

2.2 Incremental Checkpointing with Non-volatile Memory

As the scaling of computing systems, MTTF continues to decrease from days to hours, and to several minutes [11, 12]. The checkpoint interval decreases with MTTF since if the checkpoint time surpass the failure period, it means the possibility of ending up with infinite execution time. As noticed by previous work [11], frequent checkpointing is only possible by using PCRAM and is critical to benefit from incremental checkpointing schemes. As the checkpoint interval decreases, the probability of polluting a clean page becomes smaller, hence, the incremental checkpointing shows advantages over full-size checkpointing in terms of average checkpoint size.

Incremental checkpointing features user-transparency, decent portability and flexibility [3, 5, 11]. It requires a memory access monitor to track the dirty data and periodically backs them up to non-volatile memory or hard-disk. Existing researches leverage the hardware page protection mechanism to monitor write operations and back up the data in the granularity of pages. At the beginning of each checkpoint interval, all the writable pages are marked as read-only using mprotect system call. When an mprotected page is written, a page fault exception occurs and sends the SIGSEGV signal. The memory access monitor receives the signal and call the page fault exception handler to save the address of the dirty page in an external data structure. The page fault signal handler also marks the written page as writable by calling unprotect system call. At the end of the checkpoint interval, the monitor scans the data structure that tracks the dirty pages and backs up all the dirty pages. In this paper, we implement this page-protection based memory access monitor in runtime level and trigger checkpoint operation with the time interval of 5 s.

2.3 The Problem with Current Checkpoint

Generally, the overhead of incremental checkpointing is decided by the granularity of memory access monitoring [13]. The finer the granularity, the more page faults will be triggered but less data will be copied. The coarser the granularity, the less page fault occurs but more unmodified data has to be moved. The ideal case falls somewhere between which involves the least overhead to the total system performance.

The widely supported page sizes in operating system are 4K and 2M. Page-based incremental checkpointing works with 4K memory pages. 2M page is too coarse for monitoring which largely increases the checkpoint size. This will make incremental checkpoint fall back to full-size checkpoint with useless monitoring. However, current incremental checkpointing suffers from significant monitoring overhead (as well as TLB overhead) with 4K pages. Once a page is visited, a page fault occurs to handle the fault and the page will be copied at the checkpoint time. We quantify the monitoring overhead and copying overhead to clear our motivation. The rdtsc instruction is used to reckon the page-fault handler by time. We estimate the data movement from DRAM to PCRAM to simulate the copying overhead where each 8 byte atomic writes costs 600 ns. Note that both monitoring and copying overhead are generic and application-independent. Table 1 shows the average monitoring overhead and copying overhead at 4K and 2M granularity. Monitoring overhead is around 2.5x the time of copying at 4K level while copying data accounts for the most of overhead (~200x) with 2M huge page.

Table 1. Comparison of monitoring and copying overhead at 4K and 2M granularity

Granularity	Monitoring overhead	Copying overhead
4K	~100 us per 4 KByte	< 40 us per 4 KByte
2M	~0.2 us per 4 KByte	< 40 us per 4 KByte

Our experiments in Sect. 4 shows that monitoring overhead accounts for around 70 % of total checkpointing time. Therefore, a key to optimize the incremental checkpointing performance is finding a way to minimize the monitoring overhead. Specifically, a coarser-grained monitor that adapts well to the memory access pattern of applications will yield much better results. Previous work [13] has tried some intuitive methods to reduce the number of page faults during checkpoint interval, such as leaving the pages around the written pages unmonitored. However, these methods without priori information shows marginal improvement.

3 Design and Implementation

3.1 Contiguous Memory Regions to be Visited

Since no priori information is used to predict the memory regions to be visited, previous page-based incremental checkpointing has to track every write operation that trigger the page fault. We observe the memory access pattern of PARSEC and SPLASH benchmark suites at 4 K granularity and find that the size of contiguous memory regions visited by applications tends to be proportional to the size of the corresponding memory allocation. For example, if an application malloc a chunk of memory space sizing 100 pages, usually a 30 % or 50 % of pages will be contiguously visited, which are 30 and 50 pages. Given the same ratio (30 % or 50 %), for applications typically allocate thousands of pages, these numbers (30 and 50 pages in last example) will be proportionally increased. In this experiment, the total size of applications' memory allocation is divided into 2n pieces where n can be a reasonably large integer, such as 6 or 7. Each piece is regarded as "hot" piece if more than 80 % of pages inside the piece has been visited. In this definition of "hot" piece, we do not put strict limitation that all pages have to be visited in order to cover some more flexible memory access patterns.

Figure 1 shows the cumulative distribution function (CDF) and histogram of the priori information we are interested, which is the proportion between contiguous memory space visited by applications and the size of memory allocation, and their corresponding frequency. We run and statistic all applications in PARSEC and SPALSH2 to make this observation more reliable and generic. The parameter n is set to 6 so each memory allocation is divided into 64 pieces. In the histogram and CDF figures, the x-axis shows the ratio between the contiguous pieces visited by applications and the total pieces of memory allocation and y-axis shows the frequency or the proportion of corresponding ratio for memory allocations in all applications. It is noticeable that nearly 30 % has no "hot" pieces at all. Around 18 % (from 27 % to 45 %) of allocation has "hot" pieces between 30 % and 40 % of total pieces allocated while around 17 % of allocation has "hot" pieces between 60–70 % of total pieces allocated. This observation sheds new light on memory access pattern which can be utilized to appropriately loose the monitoring granularity to achieve better performance of incremental checkpointing for applications with large memory footprint.

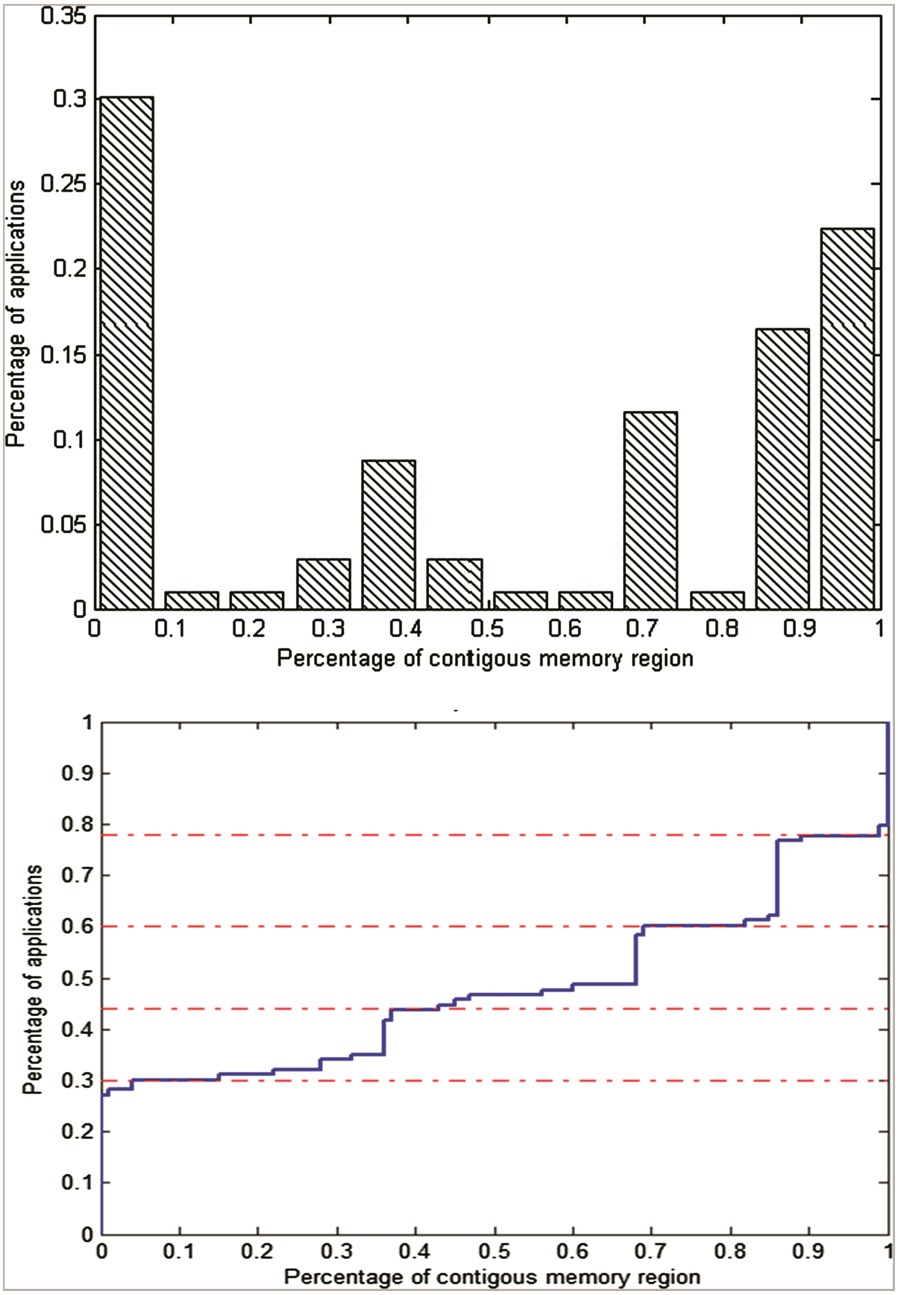

Fig. 1. Histogram and cumulative distribution function (CDF) of the ratio between contiguous memory pieces visited by applications and total allocated pieces

3.2 Application-Based Coarse-Grained Checkpoint: Loose Monitoring Granularity for "Hot" Applications

Our observation shows that applications tend to visit contiguous memory regions with its length often proportional to the size of the corresponding memory allocation. Based on this observation, we propose Application-Based Coarse-Grained Incremental Checkpointing (ACCK) that looses monitoring granularity for "hot" applications. ACCK first records of the start address and size of memory regions applied by applications. ACCK keeps track of the memory access pattern at 4K page granularity and divides the memory allocation into 2n pieces (at least one page for each piece). Here, we adopt a new definition of "hot" piece that for each piece of memory, if more than 80 % pages out of first 10 % of pages are visited, it is marked as a potential "hot" piece.

ACCK releases the monitoring granularity incrementally and gradually. Given that there are 2n pieces for each memory allocation, ACCK leaves the pieces unmonitored based on the memory access pattern at the pace of (20, 21, …, 2n − 3), (20, 21 …, 2n − 3), …, (20, 21, …, 2n − 3) pieces. Specifically, at the beginning of checkpoint, every piece is monitored. If a monitored piece becomes a potential "hot" piece, it will be left unmonitored by unprotecting all the pages inside the piece. If the following piece become a potential "hot" piece again, this piece and the following piece (a total of 21 = 2 pieces) will left unmonitored. Similarly, if following piece becomes potential "hot", ACCK leaves the following pieces unmonitored at the pace of 22, 23…until 2n − 3 pieces, then it repeat this process from the pace 20 to prevent over coarse granularity. For example, if an application visits all pieces in order, ACCK mechanism (with n = 6) releases the monitor at the pace of (1, 2, 4, 8) pieces and repeat until the end of memory regions. Figure 2 (a) shows the ACCK example for an in-order memory access pattern and Fig. 2(b) shows an ACCK example for an any-order memory access pattern. The dotted areas in Fig. 2 denote the recognized potential "hot" pieces, which trigger the following arrows. Note that the arrows in Fig. 2 does not have to occur in order.

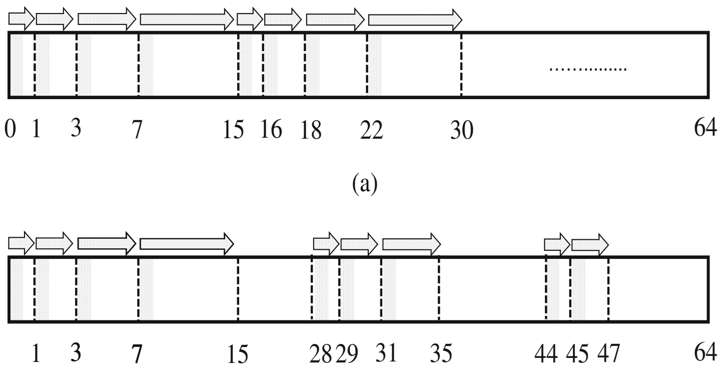

Fig. 2. Example of ACCK mechanism for in-order and any-order memory access pattern

3.3 Huge Page Support

2M huge page has a theoretical 512x performance improvement in terms of TLB performance over 4K page. However, previous researches are cautious about the use of huge pages. If huge pages are not used appropriately, a large amount of unmodified data has to be copied at the end of checkpoint interval.

In this work, we implement a kernel patch to Linux kernel 3.18 to provide merge and split operations between 4K pages and 2M huge pages. A merge operation merges 512 contiguous, 2M-alined 4K pages into a huge page (Fig. 3 top). A split operation splits a 2M huge page into 512 4K pages (Fig. 3 bottom). These functions are exposed to runtime library via system calls.

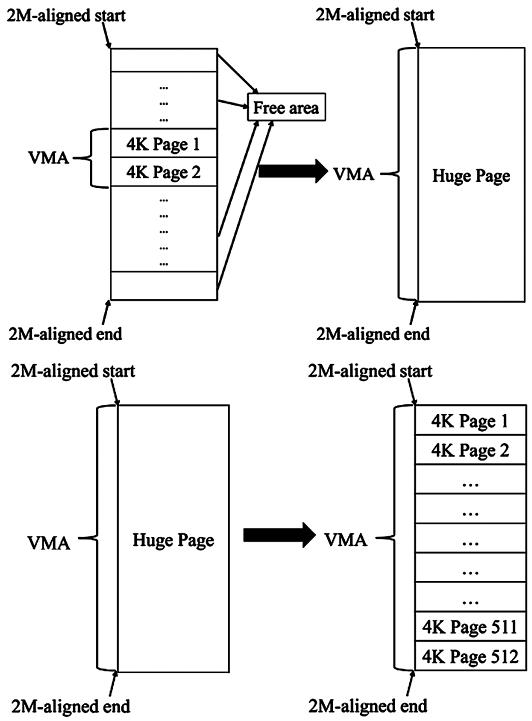

Fig. 3. An example of the merge (top) and split (bottom) operation done by modification on Linux.

In ACCK mechanism, if a large number of pieces with more than 512 4K pages in total will be unmonitored, these pages will be merged into huge pages if possible (note that huge pages to be merged into must be 2M-aligned). If the merge succeeds, ACCK turns off the page protection of the huge pages to leave it unmonitored. If the merge fails, ACCK remains to work in the traditional 4K granularity. In order to monitor in fine granularity at the beginning of checkpoint interval, the merged huge pages will be split into 4K pages at the end of checkpoint interval. The benefits of huge page comes from low TLB miss rate and penalty during runtime.

4 Experiments

This paper introduces Application-Based Coarse-Grained Incremental Checkpointing (ACCK). This section mainly evaluates and compares the performance of ACCK mechanism with the page-grained incremental checkpointing techniques.

4.1 Experimental Setup and Benchmarks

Our experiment platform is an AMD sever equipped with 2.2 GHz 12-core CPU and 16 GB of physical memory. The operating system is Linux 4.2. Our benchmarks come from PARSEC and SPLASH benchmark suite [14]. It focuses on programs from all domains, including HPC applications and server applications. The largest "native" input set is used.

4.2 Performance Metrics and Corresponding Results

Monitoring Overhead. ACCK mechanism mainly focuses on reducing the significant monitoring overhead of incremental checkpoint. Given the fact that the unit monitoring overhead is around 2.5x of unit copying overhead (application-independent), ACCK mechanism appropriately release the monitoring granularity with useful priori information. Figure 4 shows the significant performance improvement in terms of monitoring overhead in incremental checkpointing. As can be seen, ACCK mechanism lowers the monitoring overhead for all applications. The improvement can be as significant as 10x for most applications. The application freqmine is an exception with its reduction not

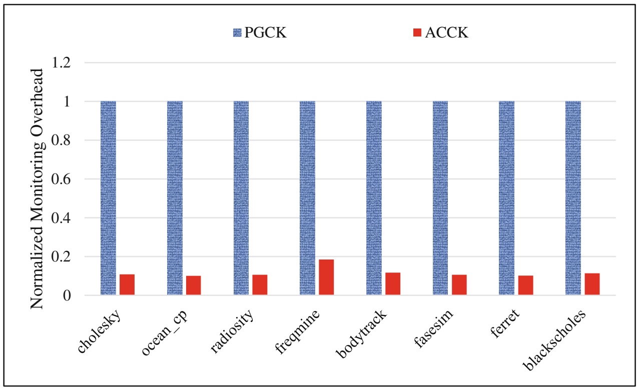

Fig. 4. Monitoring overhead

as significant as other applications. The reason behind is that there is not enough potential "hot" pieces in its memory allocation, which does not satisfy the requirements to trigger ACCK mechanism. In this case, the discrete memory access pattern will cause ACCK to fall back to the baseline incremental checkpointing. However, due to negligible management overhead of ACCK, the monitoring overhead of ACCK is always much better than traditional page-grained incremental checkpointing.

Copying Overhead. As mentioned in the previous section, the copying overhead is not as significant as monitoring overhead per memory page. ACCK mechanism releases monitoring granularity and increases copying overhead during checkpoint. However, our experimental results proves that the additional copying overhead is minimal compared to the improved monitoring performance. Figure 5 shows the total copying overhead of ACCK and the baseline incremental checkpoint. Only an average of 7.4 % more data has to be copied. Moreover, we argue that the preCopy mechanism [12] to pre-move data before checkpoint time reduces the memory pressure at checkpoint time. Therefore, the additional copying overhead can be effectively amortized.

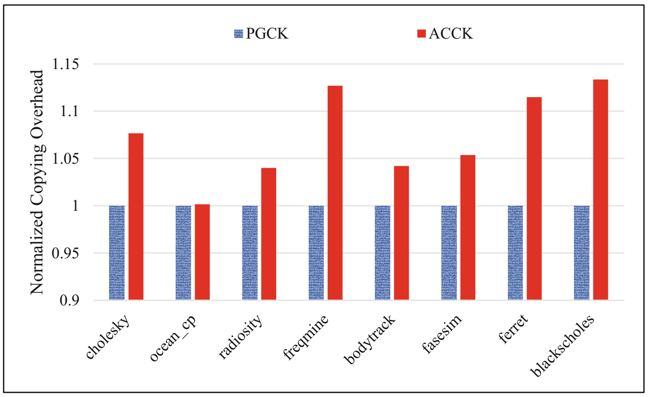

Fig. 5. Copying overhead

TLB Overhead. We also evaluate the performance of the page size adjustment. ACCK mechanism merge 4K pages into huge pages if a large number of pieces with more than 512 4K pages in total will be unmonitored. The use of huge page helps improve the TLB performance since each TLB entry maps much larger memory region with huge page. We use oprofile profiling tool to evaluate the data TLB miss using DTLB_MISS counter where the sampling count is set to 500. Figure 6 shows the normalized data TLB miss of ACCK mechanism with and without huge page support. Note that the merged pages will fall back to 4K pages at checkpoint time to maintain fine-grained monitoring at start.

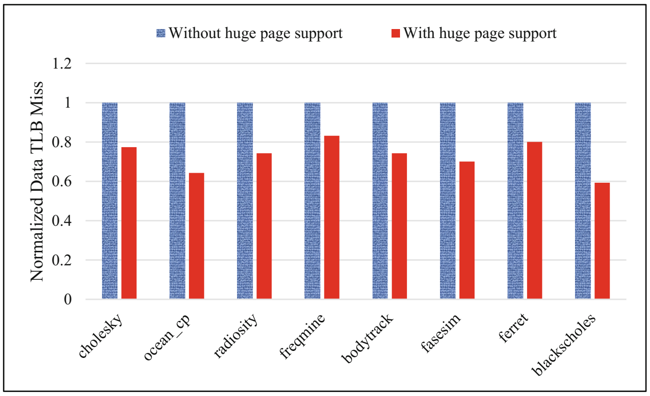

Fig. 6. Normalized data TLB miss with and without huge page support

Overall Checkpointing Performance. Generally, the overhead of incremental checkpoint consists of the monitoring overhead and the copying overhead. ACCK mechanism sacrifices the copying overhead for significant reduction in monitoring overhead and our experimental results prove it to be very effective in terms of overall checkpointing performance. Figure 6 gives the overall checkpointing performance of ACCK mechanism and the baseline incremental checkpoint. The overall checkpointing performance is defined as the reciprocal of the summation of the monitoring overhead and copying overhead. ACCK mechanism achieves 2.56x performance improvement over the baseline incremental checkpointing. It is noticeable that the performance improvement of each benchmark is pretty average, with the highest improvement 2.79x (ocean_cp) and lowest improvement 2.2x (freqmine) (Fig. 7).

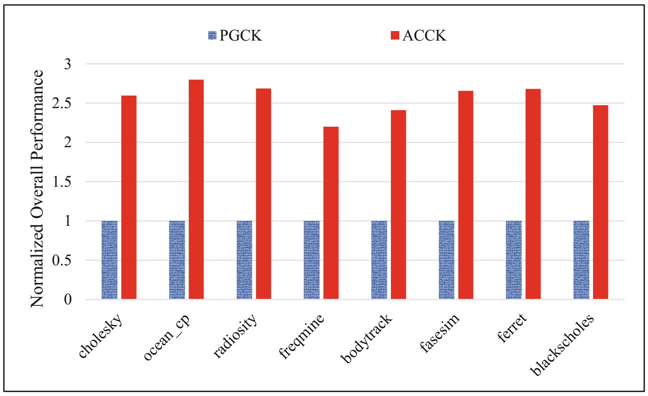

Fig. 7. Overall performance

5 Related Work

As checkpoint-restart is the commonly-used technique for fault-tolerance, the related research is abundant. Non-volatile memory sheds new light on fault tolerance. Previous work [11, 12] leverages the byte persistency of non-volatile memory to do in-memory copy to accelerate checkpointing. To address the slow write speed and limited bandwidth during checkpoint, [11] proposes a preCopy mechanism to move checkpoint data to non-volatile memory before checkpoint time to amortize the memory bandwidth pressure at checkpoint time. [12] proposes a 3D PCMDRAM design at architectural level to facilitate data movement between DRAM and PCM. These two studies both focus on hiding the long write latency of non-volatile memory.

Most of related work of reducing the checkpoint size uses the incremental checkpointing technique [15] that only saves the dirty data between two consecutive checkpoints. Since then, the hardware memory management mechanism has been leveraged to monitor the dirty data. However, incremental checkpoint technique still suffers from significant monitoring overhead and struggle with the granularity of memory access monitor. This paper propose a novel monitoring mechanism which can reduce the monitoring overhead with bearable increase in checkpoint size. Our work is orthogonal to previous studies (e.g. pre-copy mechanism [12]) and can be combined to achieve better performance.

6 Conclusion

Checkpoint-restart has been an effective mechanism to guarantee the reliability and consistency of computing systems. Our work mainly addresses the significant monitoring overhead of current incremental checkpointing technique. This paper proposes Application-Based Coarse-Grained Incremental Checkpointing (ACCK) mechanism based on non-volatile memory. We observe the memory access characteristics and find that the size of contiguous memory regions heavily visited by applications tends to be proportional to the size of allocated memory space. ACCK leverages the priori information of the memory allocation to release the memory monitoring granularity in an incremental and appropriate way. The experimental results shows that ACCK largely reduces the monitoring time and presents 2.56x overall checkpointing performance improvement. This work can be applied to frequent checkpoint of a wide range of applications and databases and can be combined with other work (e.g. pre-copy mechanism) to achieve better checkpoint performance.

Acknowledgments. This work is partially supported by National High-tech R&D Program of China (863 Program) under Grants 2012AA01A301 and 2012AA010901, by program for New Century Excellent Talents in University, by National Science Foundation (NSF) China 61272142, 61402492, 61402486, 61379146, 61272483, and by the State Key Laboratory of High-end Server and Storage Technology (2014HSSA01). The authors will gratefully acknowledge the helpful suggestions of the reviewers, which have improved the presentation.

References

1. Reed, D.: High-End Computing: The Challenge of Scale. Director's Colloquium, May 2004
2. Schroeder, B., Gibson, G.A.: Understanding failures in petascale computers. J. Phys.: Conf. Ser. **78**(1), 012022 (2007). IOP Publishing
3. Plank, J.S., Xu, J., Netzer, R.H.: Compressed differences: an algorithm for fast incremental checkpointing. Technical report, CS-95-302, University of Tennessee at Knoxville, August 1995
4. Princeton University Scalable I/O Research. A checkpointing library for Intel Paragon. http://www.cs.princeton.edu/sio/CLIP/
5. Plank, J.S., Beck, M., Kingsley, G., Li, K.: Libckpt: transparent checkpointing under unix. In: Usenix Winter Technical Conference, pp. 213–223, January 1995
6. Vaidya, N.H.: Impact of checkpoint latency on overhead ratio of a checkpointing scheme. IEEE Trans. Comput. **46**, 942–947 (1997)
7. Plank, J.S., Elwasif, W.R.: Experimental assessment of workstation failures and their impact on checkpointing systems. In: 28th International Symposium on Fault-Tolerant Computing, June 1998
8. Koltsidas, I., Pletka, R., Mueller, P., Weigold, T., Eleftheriou, E., Varsamou, M., Ntalla, A., Bougioukou, E., Palli, A., Antanokopoulos, T.: PSS: a prototype storage subsystem based on PCM. In: Proceedings of the 5th Annual Non-volatile Memories Workshop (2014)
9. Coburn, J., Caulfield, A.M., Akel, A., Grupp, L.M., Gupta, R.K., Jhala, R., Swanson, S.: NV-heaps: making persistent objects fast and safe with next-generation, non-volatile memories. ACM SIGARCH Comput. Archit. News **39**(1), 105–118 (2011). ACM
10. Bautista-Gomez, L., Tsuboi, S., Komatitsch, D., Cappello, F., Maruyama, N., Matsuoka, S.: FTI: high performance fault tolerance interface for hybrid systems. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. ACM (2011)
11. Dong, X., Muralimanohar, N., Jouppi, N., Kaufmann, R., Xie, Y.: 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. ACM (2009)
12. Kannan, S., Gavrilovska, A., Schwan, K., Milojicic, D.: Optimizing checkpoints using NVM as virtual memory. In: 2013 IEEE 27th International Symposium on Parallel and Distributed Processing (IPDPS). IEEE (2013)
13. Zhang, W.Z., et.al.: Fine-grained checkpoint based on non-volatile memory, unpublished
14. Bienia, C., Kumar, S., Singh, J.P., Li, K.: The PARSEC benchmark suite: characterization and architectural implications. In: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques. ACM (2008)
15. Sancho, J.C., Petrini, F., Johnson, G., Frachtenberg, E.: On the feasibility of incremental checkpointing for scientific computing. In: Proceedings of the 18th International Parallel and Distributed Processing Symposium. IEEE (2004)
16. Nam, H., Kim, J., Hong, S.J., Lee, S.: Probabilistic checkpointing. In: IEICE Transactions, Information and Systems, vol. E85-D, July 2002
17. Agbaria, A., Plank, J.S.: Design, implementation, and performance of checkpointing in NetSolve. In: Proceedings of the International Conference on Dependable Systems and Networks, June 2000