JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 29, 1285-1298 (2013)

DOI:10.1688/JISE.2013.29.6.13

Short Paper

A Transparent, Incremental, Concurrent Checkpoint Mechanism for Real-time and Interactive Applications*

JIANWEI LIAO
College of Computer and Information Science
Southwest University of China
Chongqing, 400715 P.R. China

TLB miss-based incremental, concurrent checkpoint mechanism for real-time and interactive applications called TIC-CKPT has been proposed, implemented and evaluated in this paper. TIC-CKPT allows setting the checkpoints overlaps with the execution of the chekcpointed processes. By resorting to tracking TLB misses to stop the first accesses to the target memory pages while saving memory address space to non-volatile storage. Meanwhile, a thread, which works in the privileged mode, copies the target pages to the designated memory buffer first, and then resumes the memory accesses. Finally the original pages in the designated memory buffer are used to construct a consistent state of the checkpointed process. From the experimental results, in contrast to a traditional concurrent checkpoint system, TIC-CKPT saves more than 2% of the checkpoint time and decreases the stopped time of the checkpointed process by around 10%. Moreover, concurrent incremental checkpointing has been designed and implemented in TIC-CKPT as well. Compared with a conventional incremental checkpoint approach, TIC-CKPT can reduce the downtime introduced by setting an incremental checkpoint to a great extent while the benchmarks keep the principle of locality.

Keywords: TLB miss, concurrent, incremental checkpoint, real-time and interactive applications

1. INTRODUCTION

The modern microprocessor and devices are susceptible to transient hardware faults due to several causes, such as the increasing number of transistors, decreasing feature sizes, reduced chip voltages and the noise margins *etc.* [1, 2]. Despite the number of those transient hardware fault is not as many as software faults, they may collapse the operating system and make the whole system go to crash with very high probability [3, 4]. From the view of software, hence, the operating systems should be reliable to recovery from such kinds of system crashes by resorting to fault-tolerant techniques. The checkpoint/ restart mechanism is a typical and effective fault-tolerant technique, which saves the state of the running process to the nonvolatile storage in the form of an image

Received June 6, 2011; revised August 27 & October 26, 2011; accepted November 11, 2011.
Communicated by Tei-Wei Kuo.

* Partial result of this research was presented at the 34th Annual IEEE International Computer Software and Applications Conference, COMPSAC 2010, Seoul, Korea, 19-23 July 2010, which was sponsored by IEEE Computer Society.

file. When the process fails caused by some external faults, such as transient hardware faults, then the stopped execution of the target process can be resumed from the latest checkpoint by reloading the state saved on the nonvolatile storage [5].

For some specified high availability and reliability applications which run in a long period, the checkpoint/restart mechanism can be employed to achieve the fault tolerance. In traditional checkpoint/restart systems, before saving the state of the checkpointed process (also called checkpointee), these checkpoint/restart techniques require stopping the checkpointee for getting its consistent state. In other words, the checkpointee cannot keep running or providing service while setting the checkpoint, and that period is referred as downtime in this paper. As we know, however, interactive applications and real-time applications need rigid timing restrictions, such as a finish time and a response time. For such applications, the downtime required while using traditional checkpoint mechanisms to set a checkpoint is always too long to be accepted. Though many checkpointing optimization techniques have been proposed to aim at these applications, the results are not general and attractive. Kai Li *et al*. [6] proposed the Low-Latency Concurrent (CLL) checkpointing, which is a technique solution designed for such applications; it enables the execution of the checkpointee overlapped with setting the checkpoints to a certain degree. However, too much memory accesses caused by the heavy operations on the checkpointee's page table reduce the benefit brought by that traditional concurrent checkpoint mechanism.

By the inspiration of CLL checkpoint system, we have proposed a new concurrent checkpoint mechanism for real-time and interactive processes in our previous work [7], which employs tracking TLB misses to block the memory accesses until the target pages are copied to the designated memory buffer during setting the checkpoint, it allows the execution of the checkpointee to overlap with the dumping of memory address space without operating on page table. Moreover, it employs two buffers to store the whole address space and the original copies of the accessed pages, and then constructs the image file of the checkpointee by using the contents in these two buffers.

For the purpose of reducing the checkpoint overhead and improving the usage of memory, based on our previous work [7], we will propose and implement an improved Transparent, Incremental, Concurrent checkpoint mechanism with a small memory buffer called TIC-CKPT in this paper. While TIC-CKPT dumps memory pages in the checkpointee's address space, the checkpointee can keep running until a memory access request (Only the first request to this memory page) is captured by tracing TLB misses, then the checkpointer(a kernel thread) copies the memory access target page to the designated memory buffer, and then unblock the memory access request. Therefore, while TIC-CKPT dumps the checkpointee's address space to disk directly, it uses the original pages stored in the designated memory buffer to construct a consistent state of the checkpointed process. Compared with CLL checkpoint system, since it does not need to operate on page table which will cause extra memory accesses, much more concurrency can be obtained by using TIC-CKPT to set checkpoints. Moreover, for the purpose of reducing the checkpoint time, TIC-CKPT supports incremental checkpointing, which means only the dirty pages after the previous checkpoint are saved to the nonvolatile storage. Compared with CRAK [8] incremental checkpoint mechanism, TIC-CKPT performs better with less checkpoint time and downtime brought by setting incremental checkpoints, while the benchmarks keep the principle of locality.

This paper is organized as follow: Section 2 introduces the background knowledge and related work. Section 3 describes the overview of the algorithm of TIC-CKPT. Section 4 presents experimental results obtained in evaluating the performance associated with TIC-CKPT. Finally, we present concluding remarks and the direction of future work.

2. BACKGROULD AND RELATED WORKS

According to the degree of checkpointing transparency, there are mainly two levels of checkpoint/restart systems [8, 9], user-level [10-12] and system-level [13, 14]. A user-level checkpoint/ restart system offers a checkpoint library, then the programmers can determine what part of the process to be checkpointed and when to set the checkpoints by calling the functions provided by the library in the application source code. A system level checkpoint/restart system is transparent to, and independent of the applications. That means neither the source code of the checkpointed application nor the compiler has to be modified for the process to be checkpointable. Due to the purpose of transparency, we just care about the system-level checkpoint mechanisms, and will refer the checkpoint/restart system as the system-level checkpoint/restart system by default in the remaining part of this paper.

2.1 Traditional Checkpointing

Lots of system-level checkpoint/restart systems have been implemented as a part of operating systems. BLCR [15] is a typical checkpoint/restart module for the Linux kernel developed and maintained by Berkeley Lab of USA; it supports for x86, ARM and PPC systems running Linux 2.6.x kernels. Kernel-based Checkpoint/Restart System [16] is an active project issued by Oren Laadan, which is a kernel-based checkpoint/restart system for the Linux kernel. In fact, traditional checkpoint systems [17-19], including those mentioned above, need to stop the checkpointed process to ensure the consistent state of the checkpointee during setting the checkpoints.

2.2 Checkpointing Optimization

To satisfy the strict timing requirements of setting checkpoints for real-time or interactive processes, several checkpoint optimization techniques have been proposed to decrease the checkpoint time. In traditional checkpoint mechanisms, reducing the checkpoint time means the downtime of the checkpointed process can be reduced as well. Decreasing the content that needed to save to the nonvolatile storage is the main direction to reduce the checkpoint time, the techniques including copy-on-write [6], diskless checkpointing [20]. Moreover, for some long time running processes which need multiple checkpoints during execution, in order to obtain an optimal interval for setting a checkpoint, Young [21] has figured out an optimal checkpoint interval, based on the assumption of Poisson failure arrivals. At last, incremental checkpointing is a wellknown technique to reduce the checkpoint time, space-efficient page-level incremental checkpointing [8, 22] and other incremental checkpointing methods [23-25] have been proposed

successively. The core idea of the incremental checkpoint mechanism is to save the modified pages (*i.e.* dirty pages) in address space of the checkpointed process from the previous checkpoint, compared with the number of all pages in the checkpointee's address space, the number of the modified pages is always smaller. Consequently, the downtime of the checkpointee while setting a checkpoint is also decreased.

2.3 CLL Concurrent Checkpointing

However, the optimized techniques mentioned in section 2.2 are solutions to the symptoms but not to the causes, they cannot reduce the downtime of the checkpointee fundamentally. As showed in our previous work [7], dumping memory address is responsible for the major part of the checkpoint time, if the execution of the checkpointee can overlap with the dumping memory address space, the downtime of the checkpointee due to setting the checkpoint can be decreased to a great extent in theory, that is motivation of the concurrent checkpoint mechanisms.

As a matter of fact, the concept of concurrent checkpointing in this paper is not a new theory, K. Li and J. S. Plank are pioneers in the study of checkpoint systems[6, 10, 20], they have proposed a low-latency, concurrent checkpoint system for parallel programs called the Concurrent Low-Latency (CLL) checkpoint system, which aims at overlapping the execution of the checkpointee with the dumping of memory address space, distinct form traditional checkpoint systems, the CLL checkpoint system works as follows:

(1) Stop the checkpointed process;
(2) Save the values of registers, thread information *etc.* to the nonvolatile storage; although they did not mention that TLB entries should to be flushed, this operation should be done before the 3th step.
(3) Turn off all the access right bits in checkpointee's page table; then resume the checkpointee;
(4) Issue copying the memory address space to a memory buffer concurrently with a kernel thread (called Copier). After copying a page, turn on the corresponding access bit. During the Copier copies the memory pages, the modified page fault handler blocks the write accesses, and invokes the Copier to copy the original target page to memory buffer first, then switch on the access right bit of the corresponding page table entry;
(5) After the Copier copies the whole address space, another kernel thread called Writer stores the data in the memory buffer to the nonvolatile storage to form an image file which contains the original copies of the write target pages.

CLL checkpoint system works quite like copy-on-write technique, it allows setting checkpoints concurrently with the checkpointee, interrupts the checkpointee only for small, fixed amounts of time, and is transparent to the checkpointee. In order to maintain consistency of the state, every write access to a page can be captured if the page has not been copied to another place because the corresponding access right bit has been turned off. After copying this write target page to the memory buffer or nonvolatile storage, the access write bit of the corresponding page table entry is restored. This means CLL checkpoint system should also operate on the page table after dumping the write target page.

After the whole checkpoint process is over, it restores all access right bits in the page table. From the description above, we can see that there are too many page table operations, which lead to extra memory references. In addition, it seems that CLL assumes all memory access requests are legal during setting checkpoints If there is an illegal write request to a read-only memory page, this request will be allowed because the access right bits in the page table are set to read-only before setting the checkpoint, operating system cannot discern whether the requests are legal or not. Of cause, it can employ two page tables or resort to hardware support, but both of them will result extra overhead, either memory accesses or the budget.

From the published literature, Li's CLL checkpoint mechanism is the sole concurrent checkpoint system that traces the modified memory pages when dumping memory address space. In this paper, we refer CLL concurrent checkpoint mechanism as the traditional concurrent checkpoint mechanism. In fact, Li's works inspired us greatly, especially, CLL checkpoint mechanism shows, the checkpointed process can keep running while dumping memory address space. Needless to say, CLL is quite suitable to set the checkpoints for real-time and interactive processes. However, the CLL checkpoint system needs too many extra memory accesses, such as setting and restoring all access right bits in the page table, which weaken the benefit brought by this kind of concurrent checkpointing directly. In addition, CLL concurrent checkpoint mechanism does not support incremental checkpointing, which is suitable for setting multiple checkpoints for the long-time running applications with quite short checkpoint time.

3. DESIGN AND IMPLEMENATION OF TIC-CKPT

A new transparent, incremental, concurrent checkpoint mechanism called TIC-CKPT in the Linux kernel will be proposed in this section, which does not require extra page table operations and supports the incremental checkpointing. Because the algorithm of the incremental checkpointing is quite different, we will present that in much more details in section 3.2.

3.1 The Architecture of TIC-CKPT

The architecture of the full checkpoint mechanism [7] is shown in Fig. 1, where checkpointer stands for the kernel thread that sets the checkpoint and works in privileged mode; Buffer B is a designated memory buffer to save the values of registers, information of the thread et cetera and the copies of memory access target pages during saving address space. Different from the CLL checkpoint mechanism, TIC-CKPT works as follows:

(1) Stop the checkpointed process by sending a "stop" signal;
(2) Copy the values of registers and thread information to the designated Buffer B rather than nonvolatile storage for decreasing the stopped time of the checkpointed process.
(3) Set the checkpoint flag to indicate that a checkpoint is being set now, and invalidate TLB entries;
(4) Resume the checkpointee by sending a "continue" signal;
(5) Save memory address space to the nonvolatile storage; meanwhile, if there is a mem-

ory access request during saving of the memory address space, since the TLB handler was modified to support concurrent checkpointing, it blocks the access request until the original target page is copied to Buffer B; then completes the loading of TLB entry; finally, the memory access proceeds as usual.

While copying the address space to the nonvolatile storage, the checkpointer scans the virtual memory areas of the checkpointee's address space, gets the virtual address of every page, and checks whether the virtual address of the page is in Buffer B or not. If not, then saves this page to the nonvolatile storage directly. If the virtual address is in Buffer B, that means there were memory accesses to this page after the starting of dumping memory address space, then the copy of the original page in Buffer B will be moved to the nonvolatile storage. Therefore, we only need a quite small memory buffer to store a list and the copies of the accessed pages rather than a big memory buffer to save a copy of the whole address space, this property enables the TIC-CKPT to be applied in memory-limited systems.

(6) Clear the checkpoint flag after the checkpointee's address space is saved to the nonvolatile storage to represent the checkpointing is completed; an image file contains a consistent state of the checkpointee is constructed and saved on the nonvolatile storage. Because both write and read requests to memory pages can be captured by tracing TLB misses [17, 26], before copying memory address space of the checkpointee, TLB should be invalidated (*i.e.* flushed), and as a result, every write or read to a page for the first time will cause a TLB miss. If the checkpoint flag is set, then every read or write request cannot be fulfilled until the original target page is copied to Buffer B. It is different from the traditional concurrent checkpoint mechanism, there are no extra memory accesses brought by the operations on page table.

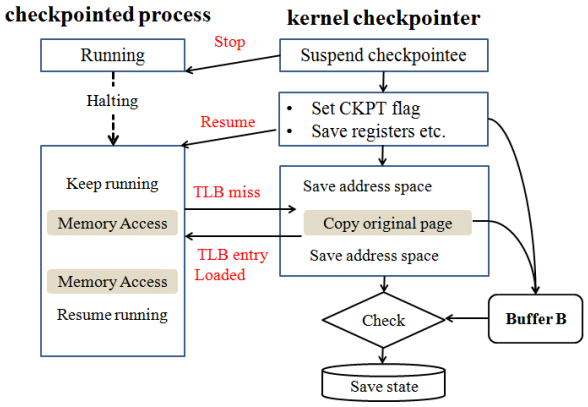


Fig. 1. Workflow of TIC-CKPT.

In TIC-CKPT, saving memory address space is being processed concurrently with the execution of the checkpointed process to a great extent. Thus, it is necessary to block the checkpointee when copying the original access target page to Buffer B before the first access request to that page. According to the locality of reference, compared with the number of pages in the whole address space, the number of the original copies of access target pages is much smaller.

3.2 Incremental Checkpointing

As mentioned in Section 2, incremental checkpointing is a widely used technique to reduce the checkpoint time, it saves the dirty pages after the previous checkpoint; there are two kinds of methods to keep track of the dirty pages. The first mechanism is using dirty bit. After setting a checkpoint, all the writable pages are cleaned as non-dirty. While the process writes the pages, operating system will set the dirty bits in corresponding page table entries. In other words, we can discern which pages are modified since the previous checkpoint, then just these pages are saved to the nonvolatile storage; the other mechanism is called bookkeeping [17], it sets all writable pages as read-only after a checkpoint, there must be a page fault exception when the page has been written. Then, the modified page fault handler inserts the address of corresponding page to a designated data structure, such as a list. At last, incremental checkpointing just need to save the pages whose addresses are in the designated data structure. However, both of mechanisms mentioned above require operating on the page table, needless to say, they bring about much longer checkpoint time. In TIC-CKPT, incremental checkpointing is also supported; it provides a mechanism like bookkeeping to track the dirty pages but without any extra operations on page table.

TIC-CKPT tracks the dirty pages by resorting to TLB modification misses (*i.e.* write violations). As mentioned before, both write and read accesses result in TLB misses, in order to distinguish them, and track the write target pages only, the modified TLB handler clears the read/write bit of the page table entry before loading it into TLB for the first time. Therefore, a write access to that 'read-only' page leads to a page fault exception, then page fault handler works as normal, finally, it calls TLB handler to load the corresponding page table entry again with the original read/write bit.

For tracing dirty pages, TLB handler has to detect that the page table entry has been loaded into TLB or not, thus a loaded list is introduced and kept by TLB handler. Before loading a page table entry into TLB cache, TLB handler checks whether the page table entry is in the list or not, if it is in, then loads that page table entry normally, otherwise, loads it after clearing the read/write bit. Fig. 2 shows how TIC-CKPT traces the dirty pages by resorting to TLB handler.

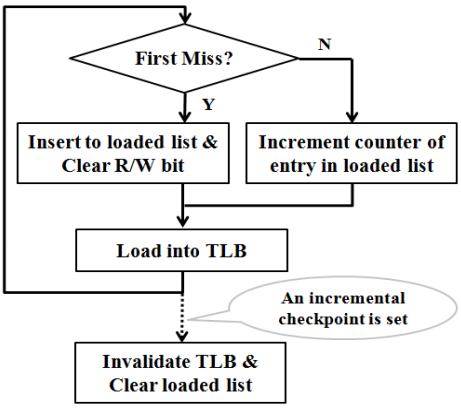


Fig. 2. Tracing dirty pages used in TIC-CKPT.

The main idea for collecting the dirty pages is to resorts to the loaded entry list, the checkpointer traverses that loaded entry list, and gathers the virtual address of the pages whose page table entries have been loaded more than once, then saves the corresponding physical pages to nonvolatile storage. At last, an incremental checkpointing image file is formed which only contains small part of pages in checkpointee's address space.

Quite different from full checkpointing workflow discussed in Section 3.1, incremental checkpointing has to invalidate all TLB entries and clear the loaded entry list before the ending of setting a checkpoint. In other words, invalidating TLB entries and clearing the loaded entry list is the last step of setting an incremental checkpoint, and the motivation of that is for supporting tracking dirty pages and making the next incremental checkpoint.

3.3 Implementation

TIC-CKPT has been implemented as a Linux module, and the target architecture is SH4 platform [27]. There are also 200 lines of source code modification in the TLB handler (the file is named tlb-sh4.c) and 8 lines patch that involves two files of the Linux kernel. Though we did not discuss the design and implementation of the restart mechanism in this paper, this functionality has been also implemented to verify the checkpoint functionality. Li's proposed CLL checkpoint system is a typical concurrent checkpoint system, for the comparison experiments, we have implemented this checkpoint system in the Linux kernel for the SH4 architecture, but we need to declare this again although it has been mentioned in Section 2.3, the experimental Linux version of CLL concurrent checkpoint system assumes all memory write requests are legal. We admit that we can use two page tables to ensure the illegal write request cannot write the read-only memory page, however, not only the degrade of concurrency due to much more comparison should be processed, but also much more modification in Linux kernel internals.

4. EXPERIMENTS AND EVALUATION

4.1 Experimental Platform and Benchmarks

In order to evaluate the performance of our proposed TIC-CKPT, we used a multiple core SH4 board as our experimental platform, called SH-4A [28], it is a 32-bit RISC microprocessor that is upward compatible with the SH-1, SH-2, SH-3, and SH-4 micro computers at instruction set code level. SH-4A has a quad-core CPU, each core equips with a maximum operating frequency of 600MHz, and 128 MB of memory. Besides, network file system has been adopted as persistent storage to save the root file system and the checkpointed image.

Before presenting the experimental results in this section, we will introduce three benchmarks used in evaluation experiments: Matrix multiplication (MAT), the size of Matrix, such as 256*256, means there are 256*256 elements in this matrix, the type of the element is double precision floating-point format; a benchmark called memperf [29], which access memory randomly; besides, according to the principles of Hartstone real-time benchmark [30], we have implemented a real-time benchmark named *rt*-bench,

which runs various periodic and sporadic tasks used in real-time DSP (Digital Signal Processing). The periodic task is the FIR (Finite Impulse Response) filter task. In addition, there are 8 sporadic tasks, such as the FFT (Fast Fourier Transform) task, the ADPCM (Adaptive Differential Pulse Code Modulation) task, the forward DCT (Discrete Cosine Transform) task and the LMS (Least Mean Square) task.

4.2 Overhead: Checkpoint Time

Fig. 3 indicates the comparison of the checkpoint times by using three checkpoint systems mentioned in Section 3.3 to set one checkpoint. We can see that checkpoint times by using both TIC-CKPT and CLL checkpoint systems are almost 20% longer than that by using non-concurrent checkpoint system.

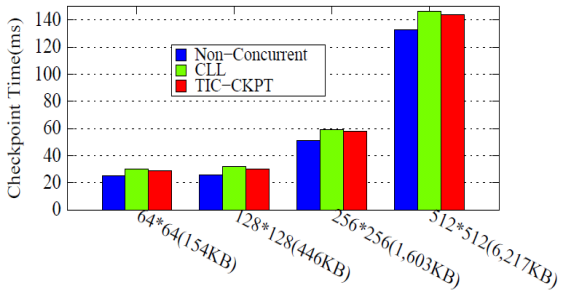


Fig. 3. Checkpoint time.

Because both TIC-CKPT and CLL checkpoint mechanisms need to check whether the pages are in the designated buffer or not before saving a page. Moreover, other operations such as invalidating TLB entries and operations on the page table take up a part of the checkpoint time. Fortunately, the focus of our work is to allow the setting of the checkpoint and running of the checkpointed process to take place concurrently; therefore, although the checkpoint times introduced by TIC-CKPT and CLL are longer than non-concurrent checkpoint system does, the absolute stopped time of the checkpointed process is much more less, such information will be presented in Section 4.3.

In addition, since the CLL checkpoint system sets all access right bits of the page table entries to be read-only before dumping memory and restores them after saving a page to non-volatile storage, it takes around 2% more time to set a checkpoint than TIC-CKPT.

4.3 Overhead: Downtime and Deadline Misses

We use the term ’downtime’ to show the stopped time of the checkpointed process while using different checkpoint mechanisms to set one checkpoint. For TIC-CKPT and CLL checkpoint mechanisms, downtime means the time needed for saving registers, thread information *etc.* and the time needed for copying the access target pages to the designated buffer before fulfilling these access requests during setting the checkpoint. In

other words, the downtimes by using them to set a checkpoint are much smaller than the checkpoint time because setting checkpoints overlaps with the execution of the checkpointee. On the other hand, for non-concurrent checkpoint mechanism, the downtime is equivalent to the checkpoint time.

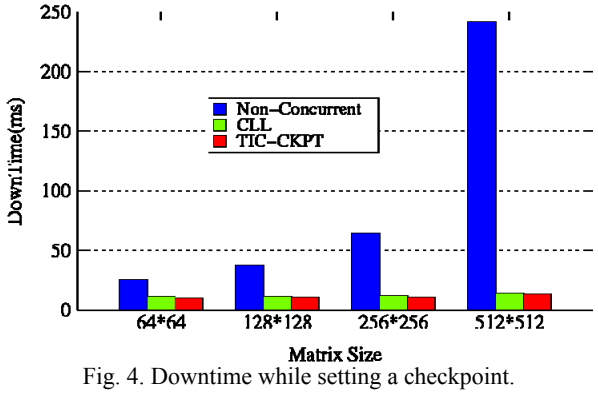


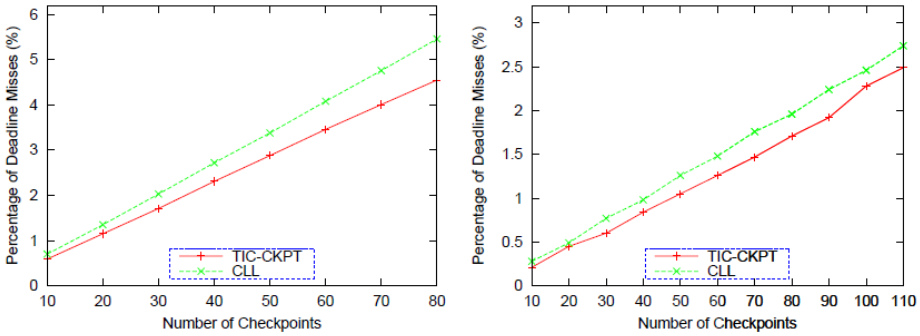
Fig. 4. Downtime while setting a checkpoint.

Fig. 4 shows that TIC-CKPT brings about the least downtime while setting a checkpoint for the matrix multiplication, in contrast to non-concurrent checkpoint system, it can reduce more than 50% downtime, especially, while the matrix size is 512*512, the reduced downtime can reach 90%. In addition, compared with CLL checkpoint system, TIC-CKPT can reduce around 10% downtime. Though 10 percent seems like a slight improvement, TIC-CKPT does not assume anything, such as all memory writes are legal during setting the checkpoint.

Moreover, to illustrate that TIC-CKPT will introduce less deadline misses by comparison to CLL checkpoint system caused by downtime; we have set various checkpoints for *rt*-bench, which runs a certain number of real-time tasks, by using both TIC-CKPT and CLL, and counted the numbers of deadline misses. We scheduled *rt*-bench to run 10000 tasks and all tasks' deadlines were configured as 1.2 times the execution time (*i.e.* 20% slack time). The first checkpoint was set at 50 ms after the start of execution, and the following checkpoints were set every 1 second if multiple ones are required. Two configurations of real-time tasks in *rt*-bench are employed in our experiments:

(1) One con_guration is that all tasks are periodic ones. Fig. 5 (a) shows the percentages of deadline misses, which is de_ned as the number of deadline misses divided by the number of total real-time tasks with various checkpoints;
(2) Another configuration is that there are 5000 periodic tasks and 5000 sporadic tasks. Fig. 5 (b) shows the percentages of deadline misses with various checkpoints.

From Figs. 5 (a) and (b), it is obvious that TIC-CKPT performs better than CLL. For instance, compared with the CLL checkpoint system, TIC-CKPT can reduce 17.4% deadline misses while there are 80 checkpoints and all tasks are periodic ones, which is shown in Fig. 5 (a). Accordingly, it is safe to conclude that TIC-CKPT introduces less deadline misses since it requires less downtime for setting checkpoints.



(a) All periodic tasks. (b) Periodic task & sporadic tasks (1:1).

Fig. 5. Percentages of deadline misses.

4.4 Incremental Checkpointing

There are two kinds of incremental checkpoint implementations in TIC-CKPT, one is called concurrent incremental checkpointing (label as TIC-CKPT-CI), that means dumping dirty pages overlaps with the execution of the checkpointed process; another one stops the checkpointed process during dumping dirty pages (labeled as TIC-CKPT). In addition, we employed CRAK [8] as our comparison counterparts to set the incremental checkpoints as well. Fig. 6 reports checkpoint times by using TIC-CKPT-CI, TIC-CKPT and CRAK to set the incremental checkpoints every 10 seconds; we record the checkpoint time for the second incremental checkpoint.

First of all, we should mention that while the benchmark is memperf, TIC-CKPT performs a little worse than CRAK, that because memperf reads and writes memory randomly, these random accesses cause much checks in TLB loaded list; for other benchmarks, TIC-CKPT reduces the checkpoint time from 2.5% to 7.1% compared with CRAK; on the other hand, Compared with TIC-CKPT and CRAK, TIC-CKPT-CI takes more than 10% time for setting an incremental checkpoint because before saving a page to the nonvolatile storage, that is because it has to check whether the page has been accessed or not after the start of dumping dirty pages.

Fig. 7 shows the downtimes while using three incremental checkpoint mechanisms. We can see that for both TIC-CKPT and CRAK, the downtime is the checkpoint time since during setting the incremental checkpoints, the checkpointed process is stopped.

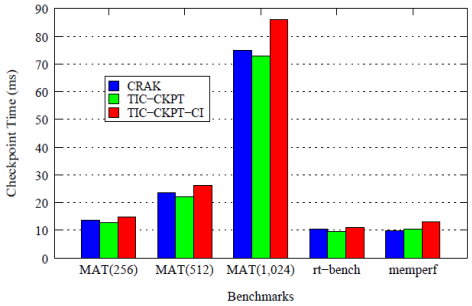


Fig. 6. Incremental checkpoint time.

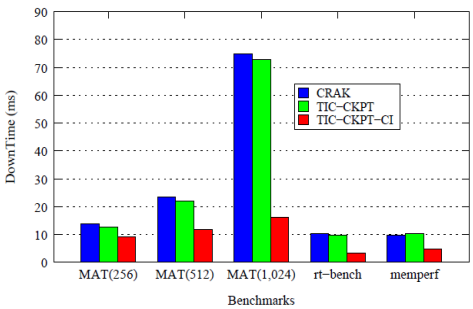


Fig. 7. Downtime with incremental checkpoint.

However, it is obvious that TIC-CKPT-CI brings about the least downtime to set the incremental checkpoints for all benchmarks on our experimental platform with quad-core CPU. Taking MAT with size 1024*1024 as an example, TIC-CKPT-CI brings about only one-fifth of the downtime brought by CRAK for setting one incremental checkpoint. We should point out that the downtime brought by TIC-CKPT-CI is equal to the checkpoint time while the target platforms are uni-core systems. Therefore, on the uni-core platforms, TIC-CKPT is a better choice in the most of cases; but for some processes which access memory randomly and run on the uni-core systems, the traditional incremental checkpoint techniques are supposed to be employed.

5. CONCLUSIONS AND FUTURE WORK

TLB miss-based concurrent, incremental checkpoint mechanism called TIC-CKPT has been designed, implemented and evaluated in this paper. This mechanism allows the checkpointed process to keep running while setting the checkpoints for it to a certain degree without any extra operations on the page table and extra hardware. Much more exactly, the most time-consuming step of setting a checkpoint (*i.e.* dumping address space) is overlapped with the running of the checkpointed process. In addition, in order to reduce the checkpoint time for setting checkpoints for long-time running processes which may need multiple checkpoints during their life time; incremental checkpointing has also been proposed and implemented in TIC-CKPT. The experimental results show that in contrast to non-concurrent checkpoint, for our selected benchmarks, it can reduce the downtime of the checkpointed process by 50%-90%. In addition, compared with the CLL checkpoint system proposed by Kai Li *et al.*, it can reduce the downtime by around 10%. For this reason, TIC-CKPT is suitable for real-time and interactive processes which have stringent timing requirements, such as a finish time or a response time.

Moreover, from the experiments on incremental checkpointing, TIC-CKPT reduces not only the checkpoint time, but also the downtime during setting incremental checkpoints. Accordingly, compared with incremental checkpoint approach implemented based on page table, TIC-CKPT performs better while the benchmarks do not access memory randomly since there are no extra operations on page table.

Though TLB misses and loads are transparent to IA-32 and IA-64 platforms, some other architectures, such as MIPS, can employ TIC-CKPT mechanism since the page table entries are loaded to TLB by operating system; Sparc and Power PC have hashed page tables that act as extended TLBs, so every TLB miss causes a fault, which is handled by the operating systems, therefore, TIC-CKPT can also be implemented in those operating systems with some minor modifications by tracing the hashed page tables on these platforms. TIC-CKPT only supports single process applications currently, it cannot set the checkpoints for multi-process applications, we will implement the checkpoint functionality to support setting the checkponts for multi-process and multi-thread applications.

ACKNOWLEDGEMENT

This work was partially supported by “Natural Science Foundation Project of CQ CSTC (No. CSTC2013JCYJA40050)” and “the Fundamental Research Funds for the

Central Universities (No. XDJK2013C025 & No. XDJK2013B005)". We would like to thank anonymous reviewers for their thorough reviews and comments to revise this paper.

REFERENCES

1. S. K. Reinhardt and S. S. Mukherjee, "Transient fault detection via simultaneous multithreading," *SIGARCH Computer Architecture News*, Vol. 28, 2000, pp. 25-36.
2. R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Transactions on Device and Materials Reliability*, Vol. 5, 2005, pp. 305-316.
3. D. Chen, S. Dharmaraja, D. Chen, L. Li, K. S. Trivedi, R. R. Some, and A. P. Nikora, "Reliability and availability analysis for the JPL remote exploration and experimentation system," in *Proceedings of International Conference on Dependable Systems and Networks*, 2002, pp. 337-344.
4. S. Borkar, "Designing reliable systems from unreliable components: The challenges of transistor variability and degradation," *IEEE Micro*, Vol. 25, 2005, pp. 10-16.
5. J. C. Sancho, F. Petrini, K. Davis, R. Gioiosa, and S. Jiang, "Current practice and a direction forward in checkpoint/restart implementations for fault tolerance," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, 2005, pp. 1-8.
6. K. Li, J. F. Naughton, and J. S. Plank, "Low-latency, concurrent checkpointing for parallel programs," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, 1994, pp. 874-879.
7. J. Liao, and Y. Ishikawa, "A new concurrent checkpoint mechanism for real-time and interactive processes," in *Proceedings of the 34th Computer Software and Applications Conference*, 2010, pp. 47-52.
8. H. Zhong and J. Nieh, "CRAK: Linux checkpoint/restart as a kernel module," Technical Report CUCS-014-01, Columbia University, November 2001.
9. S. Bhattiprolu, E. W. Biederman, S. Hallyn, and D. Lezcano, "Virtual servers and checkpoint/restart in mainstream Linux," SIGOPS Operation System Review, Vol. 42, 2008, pp. 104-113.
10. J. S. Plank, G. K. MicahBeck, and K. Li, "Libckpt: Transparent checkpointing under Unix," in *Proceedings of USENIX Conference*, 1995, pp. 213-223.
11. V. C. Zandy, http://pages.cs.wisc.edu/zandy/ckpt/README.
12. M. Bozyigit and M. Wasiq, "User-level process checkpoint and restore for migration," *SIGOPS Operation System Review*, Vol. 35, 2001, pp. 86-96.
13. SWSoft, http://download.openvz.org/doc/OpenVZ-Users-Guide.pdf.
14. F. Qin, J. Tucek, J. Sundaresan, and Y. Zhou, "Rx: treating bugs as allergies-a safe method to survive software failures," in *Proceedings of the 20th Symposium on Operating Systems Principles*, 2005, pp. 235-248.
15. P. H. Hargrove and J. C. Duell, "Berkeley lab checkpoint/restart (BLCR) for Linux clusters," *Journal of Physics: Conference Series*, Vol. 46, 2006, pp. 494 -499.
16. Kernel based checkpoint/restart, git://git.ncl.cs.columbia.edu/pub/git/linuxcr.git.
17. R. Gioiosa, J. C. Sancho, S. Jiang, and F. Petrini, "Transparent, incremental checkpointing at kernel level: a foundation for fault tolerance for parallel computers," in

Proceedings of ACM/IEEE Conference on Supercomputing, 2005, pp. 1-9.
18. E. Elnozahy and W. Zwaenepoel, "Manetho: Transparent roll back-recovery with low overhead, limited rollback, and fast output commit," *IEEE Transactions on Computers*, Vol. 41, 1992, pp. 526-531.
19. C. Wang, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive process-level live migration in HPC environments," in *Proceedings of ACM/IEEE Conference on Supercomputing*, 2008, pp. 1-12.
20. J. S. Plank, K. Li, and M. A. Puening, "Diskless checkpointing," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, 1998, pp. 972-986.
21. J. W. Young, "A first order approximation to the optimum checkpoint interval," *Communications of the ACM*, Vol. 17, 1974, pp. 530-531.
22. S. Yi, J. Heo, Y. Cho, and J. Hong, "Taking point decision mechanism for page-level incremental checkpointing based on cost analysis of process execution time," *Journal of Information Science and Engineering*, Vol. 23, 2007, pp. 1325-1337.
23. J. C. Sancho, F. Petrini, G. Johnson, and E. Frachtenberg, "On the feasibility of incremental checkpointing for scientific computing," in *Proceedings of International Parallel and Distributed Processing Symposium*, 2004.
24. N. Naksinehaboon, Y. Liu, C. B. Leangsuksun, R. Nassar, M. Paun, and S. L. Scott, "Reliability-aware approach: An incremental checkpoint/restart model in HPC environments," in *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid*, 2008, pp. 783-788.
25. J. Mehnert-Spahn and M. S. EugenFeller, "Incremental checkpointing for grids," in *Proceedings of the Linux Symposium*, 2009, pp. 201-208.
26. Y. Li and Z. Lan, "A fast restart mechanism for checkpoint/recovery protocols in networked environments," in *Proceedings of Dependable Systems and Networks With FTCS and DCC*, 2006, pp. 217-226.
27. SuperH Corporation: SH-4 CPU Core Architecture, 2002.
28. Renesas Corporation: SuperH RISC engine Family, 2009.
29. Memperf, a simple memory benchmark.
30. https://svn.mcs.anl.gov/repos/ZeptoOS/trunk/BGP/packages/zelf/src/memperf.c.
31. N. H. Weiderman and N. I. Kamenoff, "Hartstone uniprocessor benchmark: Definitions and experiments for real-time systems," *Real-Time Systems*, Vol. 4, 1992, pp. 53-382.

Jianwei Liao (廖剑伟) received M.S. degree in Computer Engineering from University of Electronic Science and Technology of China in 2006. Now, he works for the College of Computer and Information Science, Southwest University of China. He published several articles in international peer reviewed journals and IEEE conferences as the first author, his research interests are dependable operating systems and file systems.