Deterministic Execution of Multiprocessor Virtual Machines

Junkang Nong, Qingbo Wu, Yusong Tan
School of Computer, National University of Defense Technology,
Changsha 410073, Hunan, China
e-mail: {njk.jackson,wu.qingbo2008,yusong.tan}@gmail.com

***Abstract*—Deterministic execution offers a lot of benefits for debugging, fault tolerance, security of multiprocessor systems. Most previous work to address this issue either depends on custom hardware or needs to recompile the program. Some others combine the hardware and software technologies. Our goal in this work is to provide deterministic execution and repeatability of arbitrary, unmodified, multiprocessor systems without custom hardware. To this end, we propose a new abstraction of a multiprocessor virtual machine named *Deterministic Concurrency State Machine* (DCSM). With the virtual private memory model, the multiprocessor virtual machine can execute deterministically as a DCSM. With the replay of the DCSM, better debugging methods and intrusion analysis can be obtained to improve the availability and security of the whole system, not only a program. We implemented DCSM on the Kernel-based Virtual Machine (KVM) and the performance cost can be acceptable if some parameters and optimization strategies are chosen correctly based on the preliminary evaluation results.**

Keywords-availability; concurrency; deterministic execution; security

I. INTRODUCTION

Nowadays, cloud computing is accelerating the market for parallel software development. However, the concurrency in multithreaded programs brings the problem of non-determinism. This non-determinism makes a concurrency system produce different outputs, even given the same input. Such weak repeatability may not ensure that a server running in the cloud can rerun to the previous state right before the physical machine crashed. Then, the wrong results may be sent to clients.

Determinism is the foundation of replay, debugging, fault tolerance and auditing. Many intrusion analysis tools assume that the system can enforce determinism even on malicious code designed to evade analysis [1]. The replicated state machine technology [2] is also based on the assumption that the virtual machine can execute deterministically.

To address the issue of non-determinism, some work that depends on custom hardware [4-5] can obtain a good performance. For software-only solutions, some of them need to recompile the program [6, 12]. When referring to the non-determinism of the whole system, previous software-only solutions [7] primarily focus on pure record-and-replay technology, which incurs high overheads and large space costs. Other software-only technologies [15-16] are tailored to specific classes of programs, but they do not notice that the environment of the program can also induce an unexpected bug (e.g., one Mozilla bug cannot be triggered unless another program modifies the same file concurrently with Mozilla [3]).

In order to provide deterministic execution of arbitrary, unmodified, multiprocessor systems without custom hardware support, we propose *Deterministic Concurrency State Machine* (DCSM). The deterministic execution of DCSM is enforced by our modified hypervisor or virtual machine monitor which we call dVMM. This solution can ensure the repeatability of the environment-caused bug, improving the ability of debugging. Given an external input, this DCSM will make a deterministic state transition based on current execution state. Therefore, the record-and-replay technology is used on this DCSM to ensure the repeatability of the execution of a multiprocessor virtual machine.

This paper makes the following contributions. First, we propose the virtual private memory model and relative scheduling algorithm. With this model and algorithm, the multiprocessor virtual machine that encapsulates multithreaded programs can execute deterministically. As a result, the controllability can be obtained. Second, we define the Deterministic Concurrency State Machine. With this DCSM, the record-and-replay technology can be used to improve the repeatability of the whole virtual machine's execution. Meanwhile, it eliminates the large space costs due to recording the interleaving of CPUs.

The outline of this paper is as follows. In Section 2, we define the DCSM, propose the virtual private memory model and describe how the dVMM ensures the deterministic execution with a scheduling algorithm and record-and-replay technology. Section 3 describes some implementation issues. Section 4 provides some evaluation results. Section 5 discusses relevant issues. Section 6 outlines related work and Section 7 concludes.

II. DETERMINISTIC EXECUTION AND REPLAY OF MULTIPROCESSOR VIRTUAL MACHINES

A. The Problem of Non-determinism

Figure 1 [3] shows a concurrency bug in Mozilla. In this figure, if thread 2 writes the variable io_pending after thread 1 writes it, there will be an expected correct execution path. But if thread 1 writes the variable io_pending after thread 2 writes it, the expectation of the program will be violated. By default, thread 1 should initialize the variable before they execute concurrently.

This is a common concurrency bug, which makes contribution to the non-determinism of multithreaded

 ISBN: 978-1-61208-153-3

programs. If a multiprocessor system is intruded, this non-determinism will make the replay of the intrusion more difficult and result in inaccurate intrusion analysis.

Another example is shown in Figure 2, which is mentioned in DMP [4]. The table inside the figure shows the frequency of the outcome of 1000 runs on a Intel Core 2 Duo machine [4].

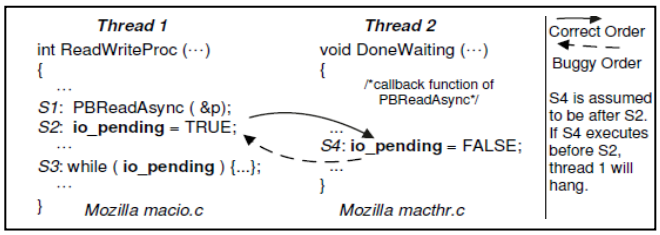

Figure 1. An order violation bug in Mozilla [3].

The result demonstrates that the underlying parallel architecture can affect the result of a system. The Symmetric MultiProcessor (SMP) model is used widely nowadays. But, Figure 2 shows the case of non-determinism in a SMP system. The non-determinism in parallel is caused by the data races of concurrent memory accesses in a SMP system. Our DCSM solution is a software-only approach to solve such a system-level issue. In addition to DCSM, external non-determinism is considered with the record-and-replay technology, which can improve the repeatability of the whole system's execution.

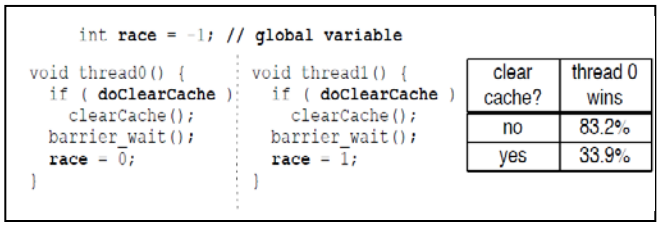

Figure 2. A simple program with a data race between two threads and runs 1000 times [4].

Next section describes the characteristics of DCSM, while its building methods are described in two following sections. The record-and-replay section specifies the method to deal with external non-determinism outside DSCM.

B. *Deterministic Concurrency State Machine*

Bocchino Jr. et al. [9] argue that parallel programming must be deterministic by default. But many programs are coded serially. They can reach parallel with the support of other tools such as compiler. Further more, people are used to thinking serially, which will probably result in buggy programs. Therefore, some measures must be taken to make the multithreaded programs execute deterministically. Those measures must also consider the environments influence on the programs. To meet these demands, a deterministic multiprocessor virtual machine is used to encapsulate the multithreaded programs and their environments. This kind of virtual machines ensures deterministic execution of the whole system. And their execution is controlled by dVMM to ensure a deterministic execution path. Such a deterministic multiprocessor virtual machine is called a *Deterministic Concurrency State Machine* (DCSM).

Figure 3 depicts the behaviors of a DCSM. A DCSM can be represented by a tuple {(V, M), I, A}, where V is the set of cpus' states, M is the set of memory states, I is the set of inputs, A is the set of actions. Given the initial state (V0, M0) and certain input, the DCSM will take a subset of actions in A and reach a deterministic state, thus produce a deterministic result. During the actions, DCSM will not receive any external inputs. The actions DCSM takes are the concurrent instructions; the size of these instructions is further determined and controlled to realize the DCSM in the following sections.

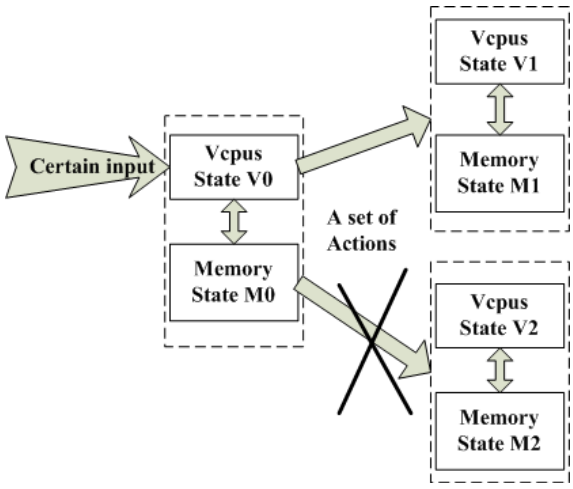

Figure 3. State transitions of a DCSM. The DCSM will deterministically take certain actions to reach state (V1, M1) if given certain input and certain initial state.

C. *Virtual Private Memory Model*

In a multiprocessor VM, if one virtual cpu (vcpu) is hung up after it acquires a lock, then other running vcpus that want to acquire this lock will waste their time in trying to get the lock. In that case, it is very difficult to control the concurrency for deterministic synchronization because of its complexity. Therefore, the basic scheduling strategy is that all vcpus in the multiprocessor VM must be running concurrently on physical cpus. Otherwise, they must be paused at the same time.

Figure 4 shows the virtual private memory model. There are two main stages-when virtual memory is created and when it is merged or synchronized.

This basic scheduling strategy makes the situation simpler. Based on that strategy, an algorithm can be designed to ensure a multiprocessor VM's deterministic execution. To be deterministic, the concurrent execution must be synchronized at some critical points. So, our algorithm is mainly based on the idea of quantum, which is composed of certain quantity of instructions. These quanta are the actions that DCSM will take to reach the next deterministic state. At

system level, data races are happened on the shared memory. So the virtual private memory model is proposed to provide each vcpu an illusion that each has its own memory. This illusion makes each vcpu executes concurrently without considering the memory interactions. As a result, without the interference from external non-determinism, an isolation of the executions of the quanta is provided before the synchronization stage. So this memory model provides the concurrent stages to fully execute and the synchronization stages to make the result of previous executions deterministic. Algorithms can use these two stages to get fully parallel execution and make the result deterministic when synchronizing.

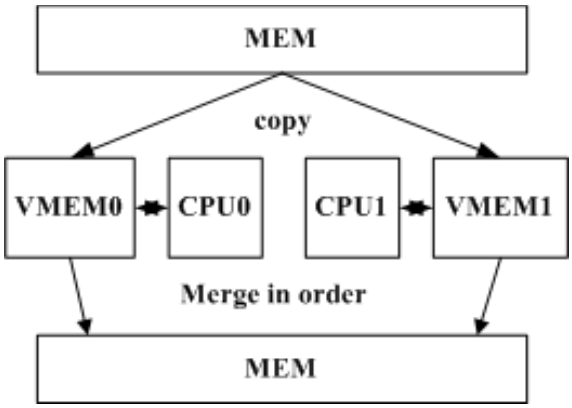

Figure 4. Virtual private memory model.

D. DMP-VPM Algorithm

At the virtualization level, the virtual private memory model ensures the virtual memory isolation of concurrent executions. According to the characteristics of virtualization level, our quantum based algorithm utilizes the virtual private memory model and privatizes the shared memory. And it synchronizes the concurrent quanta to make the result deterministically. This algorithm is called DMP-VPM (Deterministic Shared Memory Multiprocessing based on Virtual Private Memory). Figure 5 tells how a vcpu behaves in DCSM.

Figure 5 describes how the algorithm works. Each vcpu executes after obtaining its virtual private memory. When the quantum finishes, it is time to merge the private memory in order. So for the sake of the sequence guarantee at the merging or synchronization stage, the idea of token ring is utilized. The token is passed in a deterministic sequence among vcpus. A vcpu with the token has the right to merge its private memory and create a new virtual private memory. Otherwise, it must wait for its turn. In the algorithm, when a vcpu with the token wants to merge, it must make sure that the vcpu has not read the pages merged or written by the previous vcpus in this memory version. Otherwise it will create its new virtual memory and re-execute the quantum. After merging successfully, the vcpu with the token will create its new virtual memory, pass the token to the next vcpu and execute its next quantum. Such a deterministic sequence in accessing or modifying memory will result in a deterministic memory state. Note that the design of DCSM does not consider the external non-determinism which will change the execution sequence of a quantum. Under such a condition, vcpus can also reach a deterministic state.

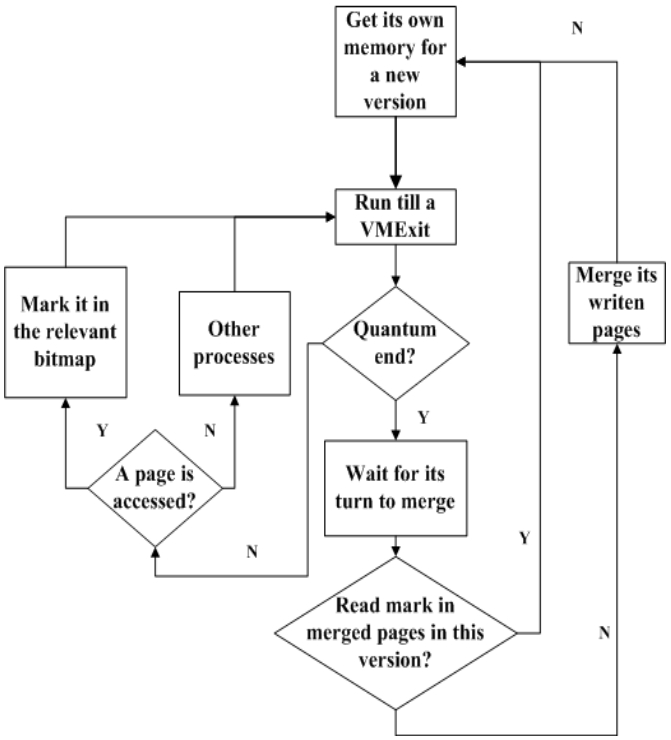

Figure 5. Each vcpu's execution diagram with virtual private memory.

E. The Record and Replay of DCSM

A deterministic executing VM can be regarded as a deterministic state machine. Because given current state and some external input, such a VM can produce a deterministic result, namely make a transition to another deterministic state. Then this VM can be replayed with the initial state and the recorded external inputs. This is the main characteristic of the DCSM. Although the record-and-replay technology has been used in many fields, the record and replay of the DCSM is kind of different.

To replay a DCSM, external non-determinism must be injected during replay at the exactly right time when the injection will not break the execution sequence of a quantum. Which vcpu needs the injection must be recorded. Some information about the non-determinism must also be recorded. When recording the DCSM, the external injections are controlled to happen at the beginning of each quantum, making the replay easier. Such an injecting method further improves the isolation between the quantum and external world and makes sure that the DCSM will not receive any input when taking actions. As a result, the quantum's result is deterministic. But some special instructions like RDTSC must be treated differently. The results of such instructions

 ISBN: 978-1-61208-153-3

and the exact occurrence time must be recorded for the replay.

III. IMPLEMENTATION ISSUES BASED ON KVM

The hardware virtual technology VT, which is developed in some Intel's cpus, makes the implementation easier and will be of great help in performance. KVM is a module in linux and makes the whole operating system a hypervisor based on VT. It utilizes the kernel's memory management mechanism to provide the guest virtual machine with a fake contiguous guest physical memory. To implement the DCSM and its record-and-replay, the interface VMEntry/VMExit is a good place for coding. The implementation framework is shown in Figure 6. According to this figure, some implementation issues are described as follows.

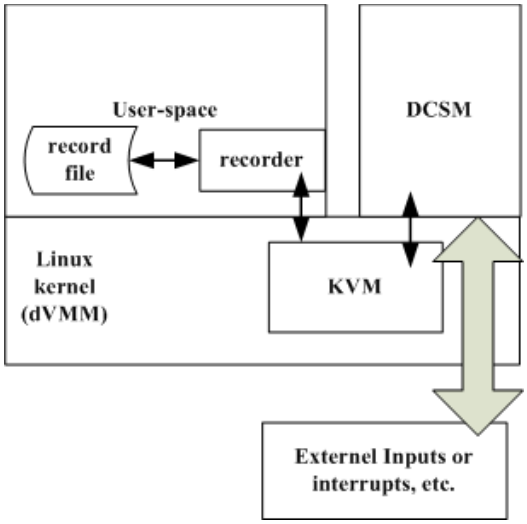

Figure 6. The framework of the dVMM.

A. Implementation of DCSM

When implementing DCSM, we have to take into account the implementation of the virtual private memory model. The implementation of that model will be described along with the DMP-VPM algorithm. The EPT (Extended Page Table) mechanism in intel's VT technology can be used to implement that model. In EPT's page table, each entry has several relevant control bits, namely readable bit, writable bit and executable bit. To provide each vcpu an isolated virtual private memory, we utilize the copy-on-write technology.

In KVM, each EPT violation will cause a VMExit which will call the relevant handler handle_ept_violation(). Then the function kvm_mmu_page_fault() is called to process this violation. In this function, the key memory process function tdp_page_fault() is called. Therefore, the process in this function can be changed to realize our goal. Note that each vcpu is designed to have its own page table. During the creation of the EPT page table, lazy allotment strategy is used. Once the guest accesses a page not present in the EPT page table, we first identify whether it is a write or a read. If it is a read, the EPT page is allotted, set as readable and not writable and the read action on this page is recorded. And if it is a write, a new host physical page is allocated and the original page's data are copied to the new page. Then dVMM will make the relevant EPT page entry redirect to the new page, set corresponding control bits and record this redirection for merging. Again, if it is a write on a page that has the EPT page present and not writable, it will be checked whether it is caused for recording. If so, dVMM will do the same thing for the write. All the records are produced during the first attempt to read or write for the sequential merging; these records are not written to the log file. After the first access to the page, the same subsequent accesses will not be interposed. So, there are only limited times of the control actions of dVMM.

B. Implementation of DCSM's record and replay

To record and replay the DCSM, some external non-determinism must be treated differently. For instructions like RDTSC, the exact logical time of the result delivery after the instruction's execution must be logged and replayed. For other external non-determinism, signals are delivered at the beginning of the quantum.

To record the exact logical time of non-determinism's occurrence, a tuple <eip, bc, ecx> is used to represent the logical time, where eip is the instruction counter, bc is the performance counter and ecx is a register used for string operations. As in figure 6, the recorder program in user space will communicate with KVM by forwarding custom commands through the ioctl() interface of the kvm device. With these commands, users can run an assigned VM as a DCSM and replay it if needed. During recording, the recorder is wakened to read the records from KVM. Then the recorder writes the records in the log file. During replay, the recorder keep extracting the records from the log file and passing them to KVM with the ioctl() interface. After receiving enough records, the assigned VM is able to run. If KVM does not have any records for replay, it will check whether the recorder has marked that all records have been sent. If not, the assigned VM will be paused until new records arrive. Otherwise, the VM continues running.

IV. EVALUATION

To get the performance evaluation of DCSM, we have to know exactly the overheads caused by the VMEXITs of the quanta for synchronization. In our virtual environment, we ran the SPLACH2 benchmark suite [17] to evaluate the design of parallel processors. For some applications, we chose input parameters to make them run for around 60 seconds so that the actual workload can be distinguished. The tests we ran were fmm, ocean, water-spatial, lu and radix. The modified virtual machine monitor KVM ran on a machine with a dual Intel Core (TM) 2 64-bit processor (2 cores total) clocked at 2.93 GHz, with 4GB of memory running linux 2.6.38.5.

Figure 7 shows the overheads of a two processor KVM guest that ran the tests in it. In the experiment, the guest's processors didn't re-execute their quanta even the quanta visited the same page. And we didn't record anything to a

 ISBN: 978-1-61208-153-3

log file. Therefore, the result mainly does not include the overheads caused by the re-execution of the quanta, nor can the record size be gained. The results of different quantum sizes as 8K, 32K, 128K were compared with that of the normal execution of the guest.

The results show that a larger quantum can reduce the VMEXITs and page faults' frequency. With a quantum size larger than 32K, the overheads are less than 3x. So a larger quantum can have a better performance in the experiment. But if quanta's re-execution is taken into account, a larger quantum will generate a longer re-execution time. In this case, the quantum size must be chosen carefully.

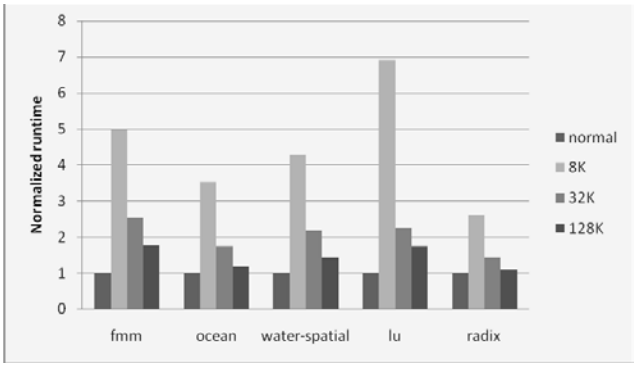

Figure 7. Overhead of DCSM for a two processor KVM guest without quanta's re-execution.

V. DISCUSSION

In the cloud, virtual machine monitors like KVM can provide users with some virtual resources based on the vast physical resources. Many servers are developed as parallel programs and can run in a virtual machine in the cloud. They may encounter an intrusion or a bug when providing service. Our solution can be applied to replay the execution and help developers fix those problems.

Our solution is quite similar to dOS [8], while dOS is implemented in an operating system and our dVMM can enforce the whole guest operating system. Based on our virtual private memory model, many other scheduling algorithms can also be applied. All evaluation experiments of dOS are done on 8-core 2.8GHz Intel Xeon E5462 machines with 10GB of RAM. Without recording the internal non-determinism due to interleaving of threads, dOS produces about 1000 times smaller logs than SMP-ReVirt [7]. The log size of dVMM depends on the communication between the virtual machine and external environment. When dealing with the entire system, dVMM can eliminate much more logs due to the interleaving of CPUs. So dVMM can have a smaller log size than SMP-ReVirt. Since dVMM have to deal with all the processes in an operating system, it will produce more logs than dOS if more processes communicate with external environment.

However, the main overhead of dVMM is due to the communication between quanta, the same as dOS. According to the evaluation of dOS, the overhead of Chromium with a scripted user session opening 5 tabs and 12 urls is about 1.7x on average. The main factors causing the overhead are the quantum size, single-stepping and the communication between quanta. Due to the cost of VMExit of each quantum for synchronization, the slowdown of dVMM is no more than 7x in our tests. But it will become much smaller if the multithreaded program has a good locality of or only a few memory accesses, as well as a suitable quantum size chosen.

There are also many optimizations that can be used to improve dVMM's performance. First, to reduce the probability of re-execution of a quantum, some methods like forward in DMP [4] can also be applied. Second, some binary translation technologies can be used to pre-process the instructions and pre-allocate some shadow pages for the vcpus to reduce more page faults and VMExits in future execution. Other optimizations can also be applied to improve dVMM's performance.

VI. RELATED WORK

At language level, parallel languages such as SHIM [10] and DPJ [9, 11] can enforce determinism, but require rewriting the code. Determinator [1] is implemented on a microkernel and proposes a new programming model. The whole new programming type may not be suitable for some common used applications and it is only implemented on a microkernel now. dOS [8] proposes Deterministic Process Groups (DPG) to ensure the concurrency determinism and a shim layer to replay DPG, which is a solution only implemented in linux. But our solution does not need a whole new programming model and supports different multithreaded programs in different operating systems.

Some hardware-based system such as DMP [4] and Calvin [5] can obtain a good performance, but require custom hardware support. DMP provides different methods to gain determinism. Some of the methods utilize the transactional memory, which is similar to our solution. However, their implementations need custom hardware support, which may not be suitable for the community hardware in the cloud. Some technologies like RCDC [12] and CoreDet [6] use a combination of hardware and software support. They not only use custom hardware, but also need software support like compliers, which forces an application to be recompiled before running. On the contrary, dVMM is a software only solution and can support arbitrary, unmodified software.

There are also many technologies focused on record and replay of a multithreaded program. Like dVMM, SMP-ReVirt [7] can replay the whole system which encapsulates multithreaded programs and their environments. Unfortunately, it has high overheads and large space costs, which is largely owing to the recording of the execution interleaving. However, DCSM does not have to record the interleaving of CPUs compared with SMP-ReVirt. PRES [13] and ODR [14] log a subset of shared memory interactions, reduce the log size, but have increased costs in replay when doing the search of the execution space. dVMM utilizes the hardware VT technology and enforces the deterministic execution with very few controls. Therefore, without logging shared memory interactions, dVMM can have much smaller

log size than SMP-ReVirt. Without searching the execution space, dVMM can perform better in replay than PRES and ODR.

VII. CONCLUSION

This paper proposed a virtual private memory model. Based on this model, DMP-VPM algorithm is introduced to control the deterministic execution of the multiprocessor virtual machine. This controlled virtual machine is called deterministic concurrency state machine. And a record-and-replay scheme for this DCSM is designed. With the internal determinism and the record of external non-determinism, the repeatability of the whole system can be ensured, providing support for debugging, intrusion analysis, etc. Without quanta's re-execution and quantum size no less than 32K, the DCSM generates overheads less than 3x. With a carefully chosen quantum size, the DCSM is supposed to have acceptable overheads.

REFERENCES

[1] A. Aviram, S. Weng, S. Hu, and B. Ford. Efficient System-Enforced Deterministic Parallelism. In OSDI. 2010, pp. 193-206.

[2] J. R. Douceur and J. Howell. Replicated Virtual Machines. Technical Report MSR TR-2005-119, Microsoft Research, Sep 2005.

[3] S. Lu, S. Park, E. Seo, and Y. Zhou. Learning from Mistakes-A Comprehensive Study on Real World Concurrency Bug Characteristics. In ASPLOS. 2008, pp. 329-339.

[4] J. Devietti, B. Lucia, L. Ceze, and M. Oskin. DMP: Deterministic Shared Memory Multiprocessing. In ASPLOS. 2009, pp. 85-96.

[5] D. Hower, P. Dudnik, D. Wood, and M. Hill. Calvin: Deterministic or Not? Free Will to Choose. In HPCA. 2011, pp. 333-334.

[6] T. Bergan, O. Anderson, J. Devietti, L. Ceze, and D. Grossman. CoreDet: A Compiler and Runtime System for Deterministic Multithreaded Execution. In ASPLOS. 2010, pp. 53-64.

[7] G. W. Dunlap, D. G. Lucchetti, P. M. Chen, and M. A. Fetterman. Execution Replay for Multiprocessor Virtual Machines. In VEE. 2008, pp. 121-130.

[8] T. Bergan, N. Hunt, L. Ceze, and S. Gribble. Deterministic Process Groups in dOS. In OSDI. 2010. pp. 177-191.

[9] R. L. Bocchino Jr., V. S. Adve, S. V. Adve, and M. Snir. Parallel Programming Must Be Deterministic by Default. In HotPar. 2009, pp. 4-4.

[10] S. A. Edwards, N. Vasudevan, and O. Tardieu. Programming shared memory multiprocessors with deterministic message-passing concurrency: Compiling SHIM to Pthreads. In DATE. 2008, pp. 1498-1503.

[11] R. Bocchino, V. Adve, D. Dig, S. Adve, S. Heumann, R. Komuravelli, J. Overbey, P. Simmons, H. Sung, and M. Vakilian. A Type and Effect System for Deterministic Parallel Java. In OOPSLA. 2009, pp. 97-116.

[12] J. Devietti, J. Nelson, T. Bergan, L. Ceze, and D. Grossman. RCDC: A Relaxed Consistency Deterministic Computer. in ASPLOS. 2011, pp. 67-78.

[13] S. Park, W. Xiong, Z. Yin, R. Kaushik, K. Lee, S. Lu, and Y. Zhou. Do You Have to Reproduce the Bug at the First Replay Attempt? - PRES: Probabilistic Replay with Execution Sketching on Multiprocessors. In SOSP. 2009, pp. 177-192.

[14] G. Altekar and I. Stoica. ODR: Output-Deterministic Replay for Multicore Debugging. In SOSP. 2009, pp. 193-206.

[15] J. Huang, P. Liu, and C. Zhang. LEAP: Lightweight Deterministic Multi-processor Replay of Concurrent Java Programs. In FSE. 2010, pp. 385-386.

[16] E. Berger, T. Yang, T. Liu, and G. Novark. Grace: Safe Multithreaded Programming for C/C++.. In OOPSLA. 2009, pp. 81-96.

[17] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In ISCA, 1995, pp. 24-36.

 ISBN: 978-1-61208-153-3