Operating System Virtualization

Priya Kumari, Neha Agrawal
Student, Department Of Information Technology
Dronacharya College Of Engg.

ABSTRACT-This paper examines potential motivations for incorporatingvirtualization support in the system software stacks of high-end capability supercomputers. We investigate virtualizing transactional memory in the contextof LogTM-SE. First, we describe an implementation of a kernelmodule in OpenSolaris that implement transactional virtualization and requires only 1120 lines of code. Second, we find that LogTMSE interacts poorly with virtual machine monitors due to a reliance on physical addresses. Third, through application tracing on real hardware and full system simulation, we show virtualizing transactions can be necessary for system stability and to support code that voluntarily context switches.We discuss our experiences in implementing functions across two major version of linux in a kernel module without the use of any kernel modification.

I. INTRODUCTION

There is an old saying that \every problem in computer sci-ence can be solved with another level of abstraction." This is commonly used in a facetious way and followed up withsomething to the e_ect of, \but performance will be horri-ble." Platform virtualization, where an extra level of ab-straction is inserted between physical hardware and the OS,indeed solves many challenging problems, but in a high per-formance computing (HPC) context the performance over-head has generally been perceived as being too high to beuseful.

Now a days virtualization has emerged as the key of technology.It introduces a level of indirection to a system to decouple application from last system.O.S virtualisation provides a full control at the level of individual process or application,which is very much useful than hardware virtualization abstraction.eg.This O.S virtualization can enable transport migration of individual application. This provides greater flexibility.We can implement O.S virtualization prototype in the entire lodable kernel module.

II. PREVIOUS WORK

Early work outlining several motivation for migrating HPC workloads to virtual machines was presented in . These include ease of management (e.g., live migration, checkpoint- restart), the ability to run custom tailored OS images (e.g., a lightweight kernel), and exposing normally privileged op- erations to unprivileged users (e.g., to load a kernel module in a guest). To reduce virtualization overheads, microprocessor vendors have recently introduced hardware to accelerate virtual mem- ory virtualization. AMD, for example, has implemented a 2-D nested page table caching scheme. This approach mitigates the potentially O(n2) memory accesses required for each guest virtual address to host physical address trans- lation, where n is the number of page table levels (n = 4 for x86-64). Intel has a similar nested paging approach called Extended Page Tables (EPT), which we evaluate in Sec- tion 4. A subsequent letter observes that the nested page table structure does not have to match the page table struc- ture exposed to the guest, and this may provide opportuni- ties for further optimization. Amazon recently began targeting HPC workloads, and now offers a high performance compute instance with 10 Gigabit Ethernet and two NVIDIA \Fermi" GPUs per node.

III. Challenges in Virtualizing TM in a Virtual

Machine Virtual machines are rapidly becoming ubiquitous and perform the same actions (context switching and paging) as an OS to virtualize the processor. In the presence of hardware transactional memory, the VMM must support virtualizing transactions to provide these services. Physical Addresses a guest OS in a virtual machine cannot correctly virtualize transactions because it lacks access to physical addresses. In a virtual machine environment, the VMM presents real addresses to the guest operating system and internally maps them onto physical addresses [30]. However, signatures operate on physical addresses. As a result, the guest OS cannot compute summary signatures after the VMM remaps a page because it knows only real addresses and not the new physical addresses of pages.

IV. VIRTUALIZING TRANSACTIONS

In this section we discuss the operations of the Transaction Virtualization Manager (TVM), a new kernel module that implements the software support for virtualizing transactional memory. It ensures that transactional semantics are enforced by the hardware when the OS reclaims a processor or context switches) occur that force a transaction to virtualize or affect a previously virtualized transaction. A thread may be in one of four transactional states:

_ N: The thread is non-transactional.

_ H: The thread is executing a hardware transaction that has not

.

memory page from a process. However, it does not currently address I/O in transactions, which require additional changes to the OS.We implement TVM in OpenSolaris [29].

Hooks in the kernel invoke TVM when virtualization events (e.g.,

been virtualized.

_ S: The thread has been suspended while executing a transaction.

_ V: The thread is executing a transaction that has been virtualized

V. Virtualizing TM in a Virtual Machine

LogTM-VSE supports virtualizing transactions both within an OS and within a VMM. This support is necessary to allow a VMM to context switch processors and swap memory without either contacting the OS to provide virtualization, or accessing internal OS data structures. While it may be possible to force transactions to abort instead, doing so makes VMM virtualization activities visible to the OS.

We designed a software architecture for simultaneous virtualization by both the VMM and a guest OS based on two principles:

1. The VMM should only be involved when it causes a transaction to virtualize.
2. The interface and interactions between the OS and VMM should be minimal.

.

These principles recognize the goal of keeping VMMs small and simple and let the OS handle the majority of virtualization events.

We duplicate TVM in both the OS and the VMM. TVMOS (TVM in the OS kernel) manages virtualization of threads due to OS events, and may execute either on bare hardware or in a virtual machine. TVMVMM manages virtualization of virtual processors due to VMM events and steps aside when the OS re-virtualizes the same transaction. Both versions of TVM compute separate summary signatures to virtualize transactions, which are merged together by TVMVMM.

VI. CONCLUSION

This paper has outlined several potential use cases for in-corporating virtualization support in the system software stacks of high-end capability supercomputers. We advocate that this will increase the exibility of these platforms sig-ni_cantly and enable new capabilities that are not possible with current _xed software stacks.

Hardware transactional memory promises to simplify multithreaded programming, which will be necessary to take advantage of future processors. To provide this benefit, though, it must support virtualizing transactions when the OS virtualizes the hardware.

REFERENCES

[1] A. Acharya and M. Raje. MAPbox: Using Parameterized
Behavior Classes to Con_ne Applications. In Proceedings of
the 2000 USENIX Security Symposium, Denver, CO, Aug.
2000.

[2] R. M. Balzer and N. M. Goldman. Mediating Connectors: A
Non-Bypassable Process Wrapping Technology. In Proceedings
of the 19th IEEE International Conference on Distributed
Computing Systems (ICDCS), Austin, TX, June 1999.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho,
R. Neugebauer, and I. P. A. War_eld. Xen and the Art of
Virtualization. In Proceedings of the 19th Symposium on
Operating System Principles (SOSP), Bolton Landing, NY,
Oct. 2003.
[4] A. Berman, V. Bourassa, and E. Selberg. TRON: Process-
Speci_c File Protection for the UNIX Operating System. In
Proceedings of 1995 USENIX Winter Conference, New
Orleans, LA, Jan. 1995.
[5] S. Bhattiprolu, E. W. Biederman, S. Hallyn, and D. Lezcano.
Virtual Servers and Checkpoint/Restart in Mainstream Linux.
SIGOPS Operating Systems Review, 42(5), July 2008.
[6] T. Boyd and P. Dasgupta. Process Migration: A Generalized
Approach using a Virtualizing Operating System. In
Proceedings of the 22nd International Conference on
Distributed Computing Systems (ICDCS), Vienna, Austria,
July 2002.
[7] How to Break Out of a Chroot Jail.
http://www.bpfh.net/simes/computing/chroot-break.html.
[8] C. Cowan, S. Beattie, G. Kroah-Hartman, C. Pu, P. Wagle,
and V. Gligor. SubDomain: Parsimonious Server Security. In
Proceedings of the 14th USENIX Systems Administration
Conference (LISA), New Orleans, LA, Mar. 2000.
[9] P. Dasgupta and T. Boyd. Virtualizing Operating System for
Seamless Disributed Environments. In Proceedings of the
IASTED International Conference on Parallel and
Distributed Computing and Systems (PDCS), Nov. 2000.
[10] T. Fraser, L. Badger, and M. Feldman. Hardening COTS
Software with Generic Software Wrappers. In Proceedings of
the IEEE Symposium on Security and Privacy, Oakland, CA,
May 1999.
[11] T. Gar_nkel. Traps and Pitfalls: Practical Problems in System
Call Interposition Based Security Tools. In Proceedings of the
Network and Distributed Systems Security Symposium, San
Diego, CA, Feb. 2003.
[12] T. Gar_nkel, B. Pfa_, and M. Rosenblum. Ostia: A Delegating
Architecture for Secure System Call Interposition. In
Proceedings of the Network and Distributed Systems Security
Symposium, San Diego, CA, Feb. 2004.
[13] D. P. Ghormley, D. Petrou, S. H. Rodriguez, and T. E.
Anderson. SLIC: An Extensibility System for Commodity
Operating Systems. In Proceedings of the 1998 USENIX
Annual Technical Conference, Berkeley, CA, June 1998.
[14] I. Goldberg, D. Wagner, R. Thomas, and E. A. Brewer. A
Secure Environment for Untrusted Helper Applications:
Con_ning the Wily Hacker. In Proceedings of the 1996
USENIX Annual Technical Conferemce, San Jose, CA, July
1996.
[15] Hackbench. http://developer.osdl.org/craiger/hackbench.
[16] S. Ioannidis and S. Bellovin. Sub-Operating Systems: A New
Approach to Application Security. Technical Report
MS-CIS-01-06, University of Pennsylvania, Feb. 2000.
[17] K. Jain and R. Sekar. User-Level Infrastructure for System
Call Interposition: A Platform for Intrusion Detection and
Con_nement. In Proceedings of the Network and Distributed
System Security Symposium, San Diego, California, Feb. 2000.
[18] M. Jones. Interposition Agents: Transparently Interposing

User Code at the System Interface. In Proceedings of the 14th
ACM Symposium on Operating Systems Principles (SOSP),
Asheville, NC, Dec. 1993.
[19] P.-H. Kamp and R. N. M. Watson. Jails: Con_ning the
Omnipotent Root. In Proceedings of the 2nd International
SANE Conference, Maastricht, The Netherlands, May 2000.
[20] E. Krell and B. Krishnamurthy. COLA: Customized
Overlaying. In Proceedings of 1992 USENIX Winter