Review, A Comparison of Software and Hardware Techniques for x86 Virtualization

Robert Hoff

October 25, 2011

1 Summary

The paper [1] compares software and hardware approaches for machine virtualisation on the x86 family of processors. In the cases considered, both the guest OS and Virtual Machine Monitor (VMM) relate to the IA-32 instruction set. Software virtualisation is achieved by interpretive binary translation, a technique established by the VMWare® Workstation and Virtual PC products. The software approach is compared to one implemented by hardware on an experimental VMM, the hardware technique uses trap-and-emulate callouts made possible by recent emergence of additional instructions in the x86 products.

It is shown that although virtualisation on the hardware interface is now possible, it is not necessarily better than pure binary translation. Hardware traps are expensive, and occur frequently on page faults, while the software approach avoids many of the exits by employing chaining optimisations between compiled code fragments. Performance tests demonstrate that the software VMM usually performs better than the HW version. The paper concludes that a chief performance bottleneck is attributed to memory management, and proposes a HW mapping scheme that could potentially improve things.

2 The Problem

Classical virtualisation [2], described as having the features of *fidelity* (identical execution compared to running directly on HW), *performance* and *safety* (VMM manages all resources) is an important problem for server consolidation between untrusted users, and is also useful as an extension of desktop computing environments. These needs arose from the emergence of the Internet and as a result of increasing commodity capabilities [3].

Previously virtualisation on the x86 processors could only be achieved on the software interface because of lack of hardware support. Commercial examples are Xen, relying on a form of para-virtualisation, and WMware that

employs binary translation. With the emergence of first-generation hardware support, the paper investigates what the performance characteristics are between current techniques and a hardware enabled VMM.

3 The Solution

In the software version, binary translation (BT) is used to overcome obstacles of classical virtualisation. The translator executes on-demand, and assumes the guest code may use the entire instruction set of the x86. The binary code is translated into compiled code fragments, that can mostly run natively. An adaptive optimisation is used to chain compiled fragments together, resulting in a decreasing proportion of translation as the guest's working set is gradually captured.

The hardware implementation uses a new IA-32 virtualisation mode for running the guest instructions directly, instructions that request or update privileged state trap execution and are passed to the VMM for emulation. Emulation and switching are expensive, the guest state is saved to a virtual machine control block, TLB flushes, and shadow page tables are updated.

4 Evaluation

Comparing the techniques described, adaptive translation generally performs better because it has ability to optimise code to avoid guest exists. It turns out that whether one uses a hardware or software VMM it is the frequency of exists that has the most costly implication on performance. This conclusion is verified in experiments, although performance is similar in many cases. In processes that require many memory accesses, such as the fork command where the guest imposes frequent context switches, software translation clearly outperforms the trap-and-emulate approach. The latter is however slightly better at executing processes that are computationally inclined. The paper recommends a hardware opportunity to improve future hardware VMM. The idea is to provide hardware assisted MMU virtualisation by hardware-walked guest page tables, this would reduce VMM intervention by translating guest addresses directly.

5 My Opinion

It is important to investigate what advantages the new hardware features of the x86 may provide. One would expect adoption of new features to be an advantage, so the experiments that were performed are counter-intuitive. The paper acknowledges that comparisons were performed on VMM entities that employ, in their entirety, either the software or the hardware interface. A useful line of investigation is of a hybrid version, for example employing

binary translation were best suited, while letting some instructions through that carry lighter trap-and-emulate responses. The VMMs compared were between a commercially evolved software version and an experimental hardware prototype. So I wonder if the prototype is sufficiently optimised to provide a fair comparison. It seems likely that proposal made for the advances in hardware technology would be beneficial, but it is not clear what the technical challenges are for implementing the scheme.

6 Questions

1. What are the engineering challenges of implementing the new hardware proposal? Has it been considered if an implementation would carry any performance drawbacks on other aspects of the processor?

2. Since the hardware VMM is a prototype, are their any known obvious features of optimisations? Considering if frequent callouts on the hardware interface can't be avoided. Is it then conclusive that even if potential optimisations were employed, performance would still favour the software VMM?

References

[1] K. Adams and O. Agesen. A comparison of software and hardware techniques for x86 virtualization. *Proceedings of the Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Oct. 2006.

[2] G.J. Popek and C.S. Kline. Formal requirements for virtualizable third generation architectures. *Cummun. ACM 17,7*, pages 412–421, 1974.

[3] M. Rosenblum. The reincarnation of virtual machines. *ACM Queue*, July/Aug. 2004.