GI-Fachgruppentreffen 2016 - Proposed Talk:

Towards Scalable Parallelization of Functional System Simulation with SimuBoost

Marc Rittinghaus Frank Bellosa

Operating Systems Group
Karlsruhe Institute of Technology (KIT)
firstName.lastName@kit.edu

Analyzing program and system behavior has become a challenging task with the advent of increasingly complex and parallel software. At the same time, hybrid main memory architectures are emerging that require an even better understanding of low-level data structure use and memory access patterns. *Functional full system simulation (FFSS)*, that is simulation at the instruction level, provides a powerful foundation to study the runtime behavior and interaction of computer architecture, operating system, and applications [1, 6, 12, 13]. Since the entire execution environment in a functional simulation is virtual, every operation[1] carried out can be easily inspected.

However, a well-known limitation with full system simulation is the *immense slowdown*. Depending on the required level of detail and the degree of instrumentation, running a workload with FFSS is up to multiple orders of magnitude slower compared to native execution. We have measured a slowdown between 30x and 200x with QEMU [2] and up to 1000x with the more accurate Simics [9] simulator. Similar numbers have also been reported by other researchers [7, 10]. That makes FFSS unpractical for long-running workloads. Along with the high slowdown comes a *loss of interactivity*. The slowdown results in high response times, which makes actively using a simulated system cumbersome and prevents successful communication of the simulated machine with non-simulated remote hosts.

SimuBoost [11] strives to close the performance gap between hardware-assisted virtualization (HAV) and FFSS to clear the way for functional simulation of interactive, network-centric, and long-running workloads. The core idea is to run the workload in a virtual machine (VM) managed by a hypervisor such as KVM [8]. At regular intervals the hypervisor takes a snapshot of the VM state (i.e., memory content, device states, etc.). The checkpoints then serve as starting points for simulations, enabling to simulate and analyze each interval simultaneously in one job per interval. By transferring jobs to multiple hosts, a parallelized and distributed simulation of the target workload can be achieved (Figure 1). Although simulations can only be started when the VM crosses an interval boundary, we could already show in a prototypical setup that the difference in execution speed between HAV and FFSS can effectively drive parallelization [5].

[1] Detailed processor- and device state models are not subject to FFSS, but they may be fed with data from functional simulation (e.g., memory trace as input for cache simulator).

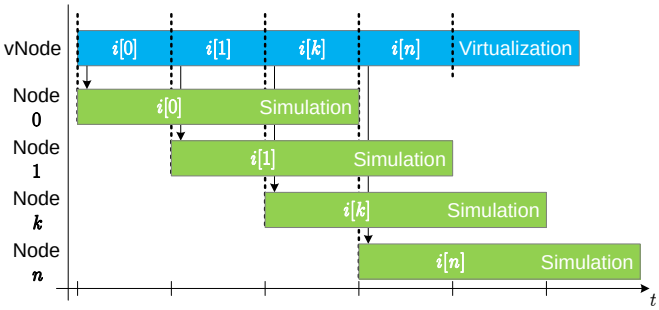

Figure 1: The workload is executed with virtualization. Checkpoints at the interval boundaries serve as starting points for parallel simulations.

Functional full system simulation can be build to always emit identical runs. Hardware-assisted virtualization, however, is subject to non-deterministic input such as erratic I/O completion timing. SimuBoost records this non-determinism and uses deterministic replay [3, 4, 14] to faithfully reproduce the execution from the HAV phase in the simulation stage. Both, checkpointing and recording non-determinism in the HAV phase, need to be geared towards low runtime overhead to (1) retain the execution speed difference, which is required for parallelization, (2) keep perturbations on the examined workload as small as possible, and (3) preserve seamless interactivity.

While we are still working on deterministic replay, our prototype already implements efficient periodic virtual machine checkpointing. Simple stop-and-copy checkpointing suspends the VM to gather a consistent snapshot of a virtual machine. This can take up to multiple seconds[2], which contradicts the requirement of low runtime overhead. In contrast, SimuBoost creates a VM checkpoint in less than 5 ms due to extensive use of asynchronous processing.

Talk. After a short introduction to the concept behind SimuBoost, the proposed talk will focus on mechanisms for lightweight periodic virtual machine checkpointing. The topics covered will include techniques such as incremental snapshots, copy-on-write (CoW), and data deduplication as well as recent hardware extensions in Intel CPUs to accelerate dirty memory page tracking. The talk will present results from our SimuBoost prototype and give an outlook on future research directions.

[2] 2 GiB RAM, running SPECjbb'05, checkpoint every 2 s

1. REFERENCES

[1] L. Albertsson et al. Using complete system simulation for temporal debugging of general purpose operating systems and workloads. MASCOT, 2000.

[2] F. Bellard. Qemu, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, 2005.

[3] T. C. Bressoud and F. B. Schneider. Hypervisor-based fault tolerance. *ACM Trans. Comput. Syst.*, 14(1):80–107, 1996.

[4] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai, and P. M. Chen. Revirt: Enabling intrusion analysis through virtual-machine logging and replay. *SIGOPS Oper. Syst. Rev.*, 36(SI):211–224, Dec. 2002.

[5] B. Eicher. Virtual machine checkpoint storage and distribution for simuboost. Master's thesis, Operating Systems Group, Karlsruhe Institute of Technology (KIT), Germany, 2015.

[6] M. Jurczyk and G. Coldwind. Identifying and exploiting windows kernel race conditions via memory access patterns. *Presented as Black Hat*, 2013.

[7] S. Kim, F. Liu, Y. Solihin, R. Iyer, L. Zhao, and W. Cohen. Accelerating full-system simulation through characterizing and predicting operating system performance. In *Performance Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Symposium on*, pages 1–11. IEEE, 2007.

[8] A. Kivity et al. kvm: the linux virtual machine monitor. volume 1. Linux Symposium, 2007.

[9] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, 2002.

[10] P. S. Magnusson, F. Larsson, A. Moestedt, B. Werner, J. Nilsson, P. Stenström, F. Lundholm, M. Karlsson, F. Dahlgren, and H. Grahn. Simics/sun4m: A virtual workstation. In *Usenix Annual Technical Conference*, pages 119–130, 1998.

[11] M. Rittinghaus, K. Miller, M. Hillenbrand, and F. Bellosa. Simuboost: Scalable parallelization of functional system simulation. In *Proceedings of the 11th International Workshop on Dynamic Analysis (WODA 2013)*, Houston, Texas, Mar. 16 2013.

[12] M. Rosenblum et al. Using the simos machine simulator to study complex computer systems. *TOMACS*, 7(1), 1997.

[13] C. Won et al. A detailed performance analysis of udp/ip, tcp/ip, and m-via network protocols using linux/simos. *High Speed Networks*, 13(3), 2004.

[14] L.-K. Yan, M. Jayachandra, M. Zhang, and H. Yin. V2e: Combining hardware virtualization and softwareemulation for transparent and extensible malware analysis. In *Proceedings of the 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments*, VEE '12, pages 227–238. ACM, 2012.