Poster Abstract

Everyone Virtualizes Everything But Time*

tiptoe.cs.uni-salzburg.at

Silviu S. Craciunas Christoph M. Kirsch Hannes Payer Harald Röck Ana Sokolova

Department of Computer Sciences
University of Salzburg, Austria

firstname.lastname@cs.uni-salzburg.at

Imagine a virtualized execution environment (VEE) that virtualizes not only the host system it runs on, even not only other systems slower than the host system, but also maintains and adjusts the exact speed at which these systems operate, in strong temporal isolation from each other, when they execute code, process I/O, and manage memory. This is what we argue for and are trying to build.

Just maintaining the exact speed at which virtualized systems operate is already a challenge but may result in an even higher proliferation of virtualization technology significantly beyond today's numbers. The potential of virtualized processors that actually perform exactly like their originals, or better, is clearly something to wish for. However, being able to adjust, at any time, the exact speed at which they operate, still in strong temporal isolation from each other, may even give rise to programming paradigms beyond putting nanosleeps of the best-guessed amount at the right place.

Here is how we believe temporal isolation can be done and even be made programmable. The core part is a real-time $O(1)$-scheduler based on the notion of variable-bandwidth servers (VBS), which we have recently developed [1]. A VBS has a bandwidth cap in percentage of CPU time, which determines the maximum CPU bandwidth available to the VBS guaranteed by the scheduler. Each VBS is in charge of exactly one scheduling task, which may be an instance of a virtual machine or just a process. New VBS and therefore new scheduling tasks are admitted to the system in constant time by checking if the sum of the bandwidth caps of the existing and new VBS remains less than or equal to the capacity of the host. In order to be scheduled, a new task chooses a so-called virtual periodic resource (λ, π), which determines the amount of time λ (called limit) the task is guaranteed to receive periodically at a given rate π (called period). Tasks can choose any virtual periodic resource and even switch subsequently from one resource to another at any time as long as the resulting CPU utilization λ over π remains less than or equal to the bandwidth cap of their associated VBS. Switching resources needs to follow a particular sequence of steps implemented by the scheduler, which guarantees that task execution resumes within at most one period of the new resource after the task initiated the switch.

A VBS-based system provides temporal isolation in the sense that the real time a given piece of code needs to execute is determined by the code itself and its inputs, independently of any other concurrent activities. Temporal isolation is even programmable with VBS since the code may choose the time (and jitter), at least within some host-dependent range.

After obtaining encouraging experimental results with our VBS implementation and simulated processes as well as when executing real virtualized code in our prototype VEE called Tiptoe [1], we are now working on integrating other subsystems for I/O and memory management with the VBS scheduler such that Tiptoe can also make workload-oriented performance guarantees (throughput and latency), not just in CPU time, but also in transmitted or allocated bytes, per unit of real time. For this purpose, we focus on subsystem implementations that can always process a given workload in CPU time that is at most linear in the size of the workload, independently of the state of the system. For example, our real-time memory management system called Compact-fit [2], can allocate and deallocate memory in constant time (unless compaction is needed for deallocation, which takes linear time in the size of the deallocated object). The VBS scheduler in combination with such subsystems may then guarantee workload-oriented performance such as I/O throughput and memory allocation rates, by using mere CPU time as common ground. A key challenge, besides the subsystem design, is to determine the exact relationship between workload size and required CPU time, which is highly implementation- and host-dependent.

*This work is supported by a 2007 IBM Faculty Award, the EU ArtistDesign Network of Excellence on Embedded Systems Design, and the Austrian Science Fund No. P18913-N15.

References

[1] CRACIUNAS, S., KIRSCH, C., PAYER, H., RÖCK, H., AND SOKOLOVA, A. Programmable temporal isolation in real-time and embedded execution environments. In *Proc. IIES* (2009), ACM.

[2] CRACIUNAS, S., KIRSCH, C., PAYER, H., SOKOLOVA, A., STADLER, H., AND STAUDINGER, R. A compacting real-time memory management system. In *Proc. ATC* (2008), USENIX.