Get What You Pay For: Providing Performance Isolation in Virtualized Execution Environments
Hannes Payer, Harald Röck, Christoph Kirsch
Department of Computer Sciences, University of Salzburg, Austria

[hpayer, hroeck, ok]@cs.uni-salzburg.at

Objective
Virtualization allows multiple systems encapsulated in so-called domains to share completely isolated from each other a single physical machine. Several companies are already taking advantage of virtualization technology in order to sell a certain amount of CPU speed and I/O capacity in terms of latency and throughput on demand to their customers. Independent of the load generated by the domains running on the system each domain has to get what its customer is paying for, not more and not less. We provide performance isolation for domain CPU speed, domain I/O throughput and latency, and time-critical applications running within a domain.

Virtual CPUs
In the Xen hypervisor the scheduling unit is called a virtual CPU (vCPU), which is an abstraction of a physical CPU. The current scheduler in Xen, called credit scheduler, is tuned for high throughput and good fairness among all active vCPUs in the system. Unfortunately, it does not provide low latency or guaranteed CPU shares [3].

EDF-vCPUs
For guaranteed CPU shares and low latency execution of vCPUs we propose the concept of an EDF-vCPU, which have a period ($\pi$) and slice ($\lambda$) as scheduling parameter. A scheduler that supports EDF-vCPUs must guarantee to run them for their respective $\lambda$ ms every $\pi$ ms.

CPU isolation inside a domain
For a virtualized OS running in a domain, a vCPU appears as a regular physical CPU core on which applications are scheduled by the OS scheduler. Therefore, adapting the hypervisor scheduler is not sufficient to provide the requested performance for an end-user application since the OS kernel introduces additional scheduling jitter. Hence, we plan to apply CPU isolation extensions (CPUISOL) to the Linux kernel in order to reduce the jitter and scheduling latencies introduced by the Linux kernel scheduler. CPUISOL isolates a CPU from the Linux scheduler, interrupts, and work-queues, while still being possible to run regular user-applications on the isolated CPU. If the isolated CPU is an EDF-vCPU, we call it an RT-vCPU since the application running on it is temporally isolated from all other applications running inside the domain (intra-domain isolation) as well as from all other domains (inter-domain isolation). Such full temporal isolation weakens to just inter-domain temporal isolation for domains that only use EDF-vCPUs but not the CPU isolation feature.

Architecture

Providing performance isolation in virtualized execution environments requires extensions across the entire system, including the hypervisor, guest kernels and driver domains. Our study is based on the open-source virtual machine monitor Xen [1]. In the system architecture of Xen, the hypervisor is the lowest layer. On top of the hypervisor run several domains, which encapsulate complete operating system instances. The main tasks of the hypervisor are domain scheduling, I/O handling, and managing memory. Some of the domains running on the hypervisor are special domains which belong to the trusted code base, e.g. a privileged domain for domain management or a driver domain where device drivers are executed.

Virtualization allows multiple systems encapsulated in so-called domains to share completely isolated from each other a single physical machine. Several companies are already taking advantage of virtualization technology in order to sell a certain amount of CPU speed and I/O capacity in terms of latency and throughput on demand to their customers.

I/O scheduling
The driver domain is similar to a network router; domains performing I/O send their requests through an event channel to the driver domain, which performs the request. We experimented with a hierarchical token bucket (HTB) traffic shaping approach in the driver domain, which regulates the packet flow from guest domains to network driver and evaluated its performance. Depending on the HTB parameters and the network back-end driver configuration (delayed copy, always copy, and never copy mode) one can trade-off network traffic throughput and driver domain CPU utilization versus network latency jitter. Moreover we aim at developing a domain bandwidth aware IRQ balancer which could distribute I/O interrupts intelligently over the available CPUs. This would optimize overall system performance and guarantee performance.

I/O Benchmark Results

References