Accurate and Efficient Process Scheduling among Virtual Machines

Hidekazu Tadokoro* Kenichi Kourai†‡ Shigeru Chiba*
*Tokyo Institute of Technology
†Kyushu Institute of Technology
‡JST, CREST

Server consolidation using virtual machines (VMs) can improve resource utilization by sharing physical resources. To share the CPU resource, the virtual machine monitor (VMM) schedules VMs and then a guest operating system (OS) in each VM schedules processes. Since each process can compete with the other processes, even in different VMs, process scheduling should consider all processes in the whole system. For example, low-importance processes in one VM should not run if high-importance processes are running in other VMs. However, it is difficult to schedule processes among VMs because a guest OS is not aware of the other OSes and the VMM is not aware of processes.

To solve this problem, we have developed a system-wide process scheduler called the *Monarch scheduler*, which enables the VMM to equally deal with processes in all VMs. The Monarch scheduler monitors the execution of processes and changes the scheduling behavior of guest OSes to achieve its custom scheduling policies. To obtain information from guest OSes such as CPU times used by processes, the Monarch scheduler uses virtual machine introspection (VMI). VMI enables examining the internal data structures inside guest OSes from the outside by using their type information.

To change scheduling policies in guest OSes, the Monarch scheduler uses direct kernel object manipulation (DKOM). DKOM is a technique that manipulates data in the OS kernel by directly modifying the kernel memory. To suspend and resume a process, the Monarch scheduler manipulates a run queue of a process scheduler in a guest OS or rewrites its state. If a process is ready in a run queue, the Monarch scheduler removes it from the run queue to suspend it. For a process waiting for an event or the currently running process, the Monarch scheduler changes its state.

We have implemented the Monarch scheduler in Xen. For ease of application and development, the Monarch scheduler is implemented as a process in domain 0, which is a privileged VM. Currently, the Monarch scheduler supports Linux 2.6 and Windows Vista as guest OSes. To examine the scheduling ability of the Monarch scheduler, we have developed two custom scheduling policies: proportional-share scheduling and idle-time scheduling.

According to our experience, it is challenging to achieve accurate and efficient process scheduling among virtual machines. First, the process times accounted in guest OSes may be inaccurate. Some operating systems performs process accounting based on timer interrupts, but timer interrupts are virtualized in VMs and may not be triggered at regular intervals. Since the Monarch scheduler uses process information inside the guest OSes, it may schedule processes based on wrong information. Second, the overhead of accessing the memory in VMs is large. To access a specified memory region in a VM, the Monarch scheduler has to look up its page table by mapping several memory pages into the address space of the process.

We are re-designing the system architecture of the Monarch scheduler so that its accuracy and efficiency are improved. For accuracy, the Monarch scheduler itself accounts process times by combining VMI with a technique called Antfarm. Antfarm enables the VMM to recognize the context switches of processes based on the CR3 register, which points to a page table in the x86 architecture. To monitor and control processes by name, the Monarch scheduler associates the value of CR3 register to a specific process by means of information obtained from guest OSes.

For efficiency, the Monarch scheduler is implemented in the VMM, not as a process in domain 0. Since the VMM does not need to map memory pages of VMs and it can directly access them, the overhead is greatly reduced. On the other hand, the safety of the Monarch scheduler lowers although the performance is improved. Slight bugs in scheduling policies may make the whole system crash. To prevent this problem, it is necessary to write scheduling policies in strongly-typed languages such as Java, Python, or Haskell.