Week 14

2019710305

이호영

1. mClock: Handling Throughput Variability for Hypervisor IO Scheduling (OSDI'10)

Virtualized servers run different sets of virtual machines (VMs), from interactive desktops to test and development environments, and even to batch workloads. The hypervisor multiplexes underlying hardware resources between VMs and uses resource management controls to provide the desired level of isolation. Existing methods provide a lot of tuning for allocating CPU and memory to VMs, but support for IO resource allocation control is quite limited. Hypervisor's IO resource management introduces significant new challenges and requires more control than a regular operating system. This white paper introduces a new algorithm for allocating IO resources in the hypervisor. Our algorithm, mClock, supports proportional shared fairness based on the minimum reservation and maximum limit on the IO allocation of a VM. Introduces mClock design and prototype implementation within the VMware ESX server hypervisor. Our results indicate that this rich QoS control is very effective in separating VM performance and providing better application latency. It also demonstrates the adaptation of mClock (dmClock) to distributed storage environments where storage is commonly provided by multiple nodes..

2. FlashFQ: A Fair Queueing I/O Scheduler for Flash-Based SSDs (ATC'13)

FlashFQ is a new I/O scheduler that ensures high responsiveness and fairness on systems that use Flash-based storage. So, let's start presenting. Timeslice-based schedulers may exhibit poor responsiveness, particularly when there are large number of co-running tasks. And since timeslice-based scheduler dispatch I/O with serialization, it is hard to exploit parallelism of flash storage device. On the other hand, a fair queuing scheduler can guarantee fairness better than timeslice-based schedulers because fair queuing schedules requests in terms of weight in finer-grained time units. FlashFQ is implemented based on SFQ (D) scheduler. SFQ (D) operates as SFQ, but it can simultaneously process D requests in parallel. FlashFQ is good scheme because it provides fast responsiveness and fairness. However, I think FlashFQ cannot support journaling thread or flush thread's fairness. So, I propose FlashFQ to support writer thread.