Architectural Support for Preemptive RAID Schedulers

Zoran Dimitrijević, Raju Rangaswami, and Edward Chang
University of California, Santa Barbara
{zoran@cs,raju@cs,echang@ece}.ucsb.edu

1. INTRODUCTION

Emerging applications such as video surveillance, large-scale sensor networks, and virtual reality require high-capacity, high-bandwidth RAID storage to support high-volume IOs. In addition to high throughput performance, increasing numbers of applications require real-time data delivery or short response time.

What is the worst-case access time, and how can it be mitigated? On an idle disk, the access time is composed of a seek and a rotational delay. When the disk is servicing a non-preemptible IO, a new IO must wait at least until after the on-going IO has been completed. In Semi-preemptible IO [1], we investigated the preemptibility of disk IOs. However, whenever the disk scheduler decides to preempt a sequential disk access, the preemption leads to additional seek overhead. In our current work [2], we investigate a class of preemptive schedulers for QoS-aware RAID systems.

Simple priority-based scheduling, if not performed carefully, can incur excessive overhead and thereby degrade the disk throughput unnecessarily. Figure 1 depicts a large sequential disk access, which can be serviced either using multiple non-preemptible low-level disk IOs, or using a single semi-preemptible IO. For example, the new IO can arrive at either time t_1 or t_2. Now, a simple priority-based scheduler will preempt the long sequential write access (and incur a preemption overhead) regardless of whether the new IO arrives at time t_1 or t_2. However, preempting the ongoing sequential IO at t_2 may not be profitable, since the ongoing IO is nearly completed. Such a preemption is likely to be counter-productive—not gaining much in response time, but incurring preemption overhead. Our Praid [2] scheme is able to discern whether and when a preemption should take place.

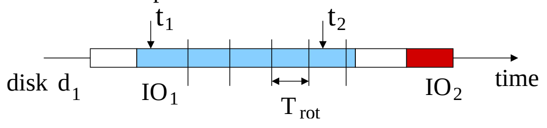

Figure 1: Sequential disk access.

2. PREEMPTION DECISIONS

In $Praid$ [2], we present *preemption* mechanisms to allow an ongoing IO to be preempted at optimal points and *resumption* mechanisms to resume a preempted IO on the same or a different disk. In addition to the mechanisms, we propose scheduling policies to decide whether and when to preempt, for maximizing the *yield*, or the total value, of the schedule. Since the yield of an IO is application- and user-defined, our scheduler maps external value propositions to internal yields, producing a schedule that can maximize total external value for all IOs, pending and current.

JIT-preemption is a method for preempting an ongoing semi-preemptible IO at the points that minimize the rotational delay at the destination track (for the higher-priority IO which is serviced next). Figure 1 depicts these possible JIT-preemption points (which are roughly one disk rotation apart). If IO_1 is preempted between these points, the resulting service time for IO_2 would be exactly the same as if the preemption is delayed until the next possible JIT-preemption point. For example, if the disk services a large write IO that is already buffered in a non-volatile RAID buffer, *preempt-always* strategy may improve read response time substantially.

JIT-migration is a method for the preemption and migration of an ongoing semi-preemptible IO in a fashion that minimizes the service time for the preempted IO. The ongoing IO is preempted only when the destination disk for the migration is ready to perform data-transfer for the remaining portion of the IO.

3. PRAID SYSTEM ARCHITECTURE

External IOs are issued by the IO scheduler external to the RAID system (for example, the operating system's disk scheduler). These IOs are tagged with their QoS requirements, so that the RAID scheduler can optimize their scheduling. We have extended a Linux kernel to enable such an IO interface [3]. The yield functions are attached to each QoS class and specify the QoS value added to the system upon the completion of an external IO.

Internal IOs are IOs which reside in the scheduling queues of individual disks in a RAID. They are tagged with internally generated QoS-value functions, and serviced using *Semi-preemptible IO*. When an internal IO is serviced, its completion yields some QoS value. However, it is hard to estimate this value. First, external QoS value is generated only after the completion of the last internal IO due for a parent external IO. Second, when performing write-back operations for buffered write IOs, their external QoS value has been already harvested. However, not servicing these internal IOs implies that servicing future write IOs will suffer when the write buffer gets filled up. Third, internally generated IOs must be serviced although their completion does not yield any additional external QoS value.

Whenever a new IO arrives, the scheduler checks whether preempting the ongoing IO, servicing the new IO, and immediately resuming the preempted IO, offers a better average yield than the one obtained without preemption. We further investigate QoS-aware preemptive schedulers in our technical report [2].

4. CONCLUSION

The specific contributions of our approach are as follows:

- *Preemption mechanisms.* We introduce two methods to preempt disk IOs in RAID systems—JIT-preemption and JIT-migration.
- *Preemptible RAID policies.* We propose scheduling methods which aim to maximize the total QoS value and use greedy approaches to decide whether the preemption is beneficial.
- *System architecture of the preemptible RAID system.* We present an architecture for QoS-aware RAID systems [2] and implement a simulator (PraidSim).

5. REFERENCES

[1] Z. Dimitrijevic, R. Rangaswami, and E. Chang. Design and implementation of Semi-preemptible IO. *Proceeding of Usenix FAST*, March 2003.

[2] Z. Dimitrijevic, R. Rangaswami, and E. Chang. Preemptive RAID scheduling. *UCSB Technical Report*, March 2004.

[3] Z. Dimitrijevic, R. Rangaswami, M. Sang, K. Ramachandran, and E. Chang. UCSB-IO: Linux kernel extensions for QoS disk access. *http://www.cs.ucsb.edu/∼zoran/ucsb-io*, 2003.