Adaptive Context Switch for Very Fast Block Device

Jongmin Gim Kwangho Lee Youjip Won
Dept. of Electrical and Computer Engineering
Hanyang University, Korea
$\{jmkim|kanlee|yjwon\}$@ece.hanyang.ac.kr

Abstract

Traditional block device (HDD) has large and unpredictable access time, from 1ms to 17ms. This characteristic imposes operating system to perform context switch to increase CPU utilization when a process requests I/O. In order to maximize CPU utilization, operating system performs context switch with adequate scheduling mechanism. Meanwhile, device interrupts to operating system when it is ready to transfer target data blocks. Before I/O procedure is progressed, one more context switch from current process context to device context has to be made which considered as overhead. In response time's point of view, this mechanism sacrifices minimum two times of context switch overhead. Overheads of context switch include loading a new working set for next process to cache, and flushing TLB. To measure cache effect in context switch, we utilize lat_ctx in $Lmbench$. The result shows that required time for ordinary context switch is under $1\mu s$, however, context switch with 2MB data loading to data cache spends from $180\mu s$ to $330\mu s$ in Intel Core 2 Duo E6550 2.33GHz Processor. Modern block devices based on memory technology (SSD, flash memory, non-volatile memory and etc) have different I/O characteristics against hard disk drive. SSD based on Flash memory requires $9\mu s \sim 42\mu s$, and FRAM at most requires 90ns $\sim$ 110ns access time. It shows a possibility that access time can be shorter than total context switch overhead, which lowers the efficiency of CPU utilization when context switch is over.

We concentrate on the point that I/O response time of fast block device based on memory can be predictable since it does not require any mechanical moving, e.g., seek, rotational delay. We suggest adaptive context switch scheme determining whether context switch is necessary or not, when process requests I/O. It performs polling in current process state, and decides whether to omit device context switch in order to reduce response time. The decision is made when overheads are smaller than that of performing switch. To achieve our goal, we (i) analyze characteristics of I/O device and context switch process, (ii) design adaptive context switch model, (iii) develop prediction algorithms which can select whether performing context switch or not, and (iv) reduce I/O response time.

The essential part of implementing adaptive context switch model is building pseudo device that uses memory and modifies parts of block I/O codes, $make\_request$ and $blkdev$, in Linux kernel 2.6.24.7. We format pseudo device by EXT2, and put a 100MB dummy file in it. Performance evaluation is done by the mean time of reading whole file measured with 1000 times. We do not take account of DMA since pseudo-device uses only the memory, no other peripherals. Fig. 1 shows the percentage of the performance enhancement of adaptive context switch scheme compared to original context switch model. ”anti” and ”noop” represent the anticipatory and noop I/O scheduling algorithm of Linux kernel, respectively. ”random” represents a reading 4KB block-sized data from each first and last position of the dummy file, and ”sequential” represents reading 4KB block-sized data sequentially from the beginning of the dummy file. Adaptive context switch scheme is tested by noop I/O scheduling, but original model is tested by anticipatory and noop. Most I/O scheduling consider disk geometry to reduce head moving even though it is not proper to block device based on memory. Fig. 1 shows that random data test gives 2% of improvement where as sequential improvement of 5.4% when original and adaptive context switch use noop I/O scheduling algorithm(noop vs noop). However the enhancement is increased up to 9% when original context switch use anticipatory I/O scheduling, whereas adaptive context switch uses noop(anti vs noop). In another experiment, we set single process which requests I/O and 9 processes which only use arithmetic operations. The results of read performance show that response time is reduced up to 16.7% even though there is only 1 process which uses device queue. We conclude that cache replacement in context switch can lead serious performance degradation, and adaptive context switch can offer significant performance improvement in response time, booting time which includes loading daemons, and CPU utilization.

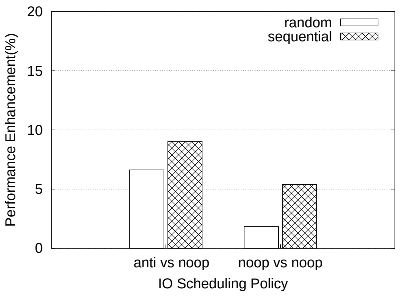

Figure 1. random and sequential read test