Chapter 6

Application of both Temporal and Spatial Localities in the Management of Kernel Buffer Cache

Song Jiang
Wayne State University, USA

ABSTRACT

As the hard disk remains as the mainstream on-line storage device, it continues to be the performance bottleneck of data-intensive applications. One of existing most effective solutions to ameliorate the bottleneck is to use the buffer cache in the OS kernel to achieve two objectives: reduction of direct access of on-disk data and improvement of disk performance. These two objectives can be achieved by applying both temporal locality and spatial locality in the management of the buffer cache. Traditionally only temporal locality is exploited for the purpose, and spatial locality, which refers to the on-disk sequentiality of requested blocks, is largely ignored. As the throughput of access of sequentially-placed disk blocks can be an order of magnitude higher than that of access to randomly-placed blocks, the missing of spatial locality in the buffer management can cause the performance of applications without dominant sequential accesses to be seriously degraded. In the chapter, we introduce a state-of-the-art technique that seamlessly combines these two locality properties embedded in the data access patterns into the management of the kernel buffer cache management. After elaboration on why the spatial locality is needed in addition to the temporal locality, we detail a framework, DULO (DUal LOcality), in which these two properties are taken account of simultaneously. A prototype implementation of DULO in the Linux kernel as well as some experiment results are presented, showing that DULO can significantly increases disk I/O throughput for real-world applications such as Web server, TPC benchmark, file system benchmark, and scientific programs. It reduces their execution times by as much as 53%. We conclude the chapter by identifying and encouraging a new direction for research and practice on the improvement of disk I/O performance, which is to expose more disk-specific data layout and access patterns to the upper-level system software for disk-oriented policies.

DOI: 10.4018/978-1-60566-850-5.ch006
INTRODUCTION

The hard drive is the most commonly used secondary storage device supporting file accesses and virtual memory paging. While its capacity growth pleasantly matches the rapidly increasing data storage demand, its electromechanical nature causes its performance improvements to lag painfully far behind processor speed progress. It is apparent that the disk bottleneck effect is worsening in modern computer systems, while the role of the hard disk as dominant storage device will not change in the foreseeable future, and the amount of disk data requested by applications continues to increase.

The performance of a disk is constrained by its mechanical operations, including disk platter rotation (spinning) and disk arm movement (seeking). A disk head has to be on the right track through seeking and on the right sector through spinning for reading/writing its desired data. Between the two moving parts of a disk drive affecting its performance, the disk arm is its Achilles’ Heel. This is because an actuator has to move the arm accurately to the desired track through a series of actions including acceleration, coast, deceleration, and settle. As an example, for a typical high performance drive of 10,000 RPM, average seek time is 6.5 milliseconds, while its average rotation time is 3 milliseconds. Thus, accessing of a stream of sequential blocks on the same track achieves a much higher disk throughput than that accessing of several random blocks does.

In the current practice, there are several major efforts in parallel to break the disk bottleneck. One effort is to reduce disk accesses through memory caching. By using replacement algorithms to exploit the temporal locality of data accesses, where data are likely to be re-accessed in the near future after they are accessed, requests for on-disk data can be satisfied without actually being passed to disk. To minimize disk activities in the number of requested blocks, all current replacement algorithms are designed by choosing block miss reduction as the sole objective. However, this can be a misleading metric that may not accurately reflect real system performance. For example, requesting ten sequential disk blocks can be completed much faster than requesting three random disk blocks, where disk seeking is involved. To improve real system performance, spatial locality, a factor that can make a difference as large as an order of magnitude in disk performance, must be considered. However, spatial locality is unfortunately ignored in current buffer cache managements. In the discussion of this chapter, spatial locality specifically refers to the sequentiality of the disk placements of the continuously requested blocks.

Another effort to break the disk bottleneck is reducing disk arm seeks through I/O request scheduling. I/O scheduler reorders pending requests in a block device’s request queue into a dispatching order that results in minimal seeks and thereafter maximal global disk throughput. Example schedulers include Shortest-Seek-Time-First (SSTF), C-SCAN, as well as the Deadline and Anticipatory I/O schedulers (Iyer et al. 2001) adopted in the current Linux kernels.

The third effort is prefetching. A prefetching manager predicts future request patterns associated with a file opened by a process. If a sequential access pattern is detected, then the prefetching manager issues requests for the blocks following the current on-demand block on behalf of the process. Because a file is usually contiguously allocated on disk, these prefetching requests can be fulfilled quickly with few disk seeks.

While I/O scheduling and prefetching can effectively exploit spatial locality and dramatically improve disk throughput for workloads with dominant sequential accesses, their ability to deal with workloads mixed with sequential and random data accesses, such as those in Web services, databases, and scientific computing applications, is very limited. This is because these two strategies are positioned at a level lower than the buffer cache. While the buffer cache receives