Disk Built-in Caches: Evaluation on System Performance

Yingwu Zhu
Department of ECECS
University of Cincinnati
Cincinnati, OH 45221, USA
zhuy@ececs.uc.edu

Yiming Hu
Department of ECECS
University of Cincinnati
Cincinnati, OH 45221, USA
yhu@ececs.uc.edu

Abstract

Disk drive manufacturers are putting increasingly larger built-in caches into disk drives. Today, 2 MB buffers are common on low-end retail IDE/ATA drives, and some SCSI drives are now available with 16 MB. However, few published data are available to demonstrate that such large built-in caches can noticeably improve overall system performance. In this paper, we investigated the impact of the disk built-in cache on file system response time when the file system buffer cache becomes larger. Via detailed file system and disk system simulation, we arrive at three main conclusions: (1) With a reasonably-sized file system buffer cache (16 MB or more), there is very little performance benefit of using a built-in cache larger than 512 KB. (2) As a readahead buffer, the disk built-in cache provides noticeable performance improvements for workloads with read sequentiality, but has little positive effect on performance if there are more concurrent sequential workloads than cache segments. (3) As a writing cache, it also has some positive effects on some workloads, at the cost of reducing reliability. The disk drive industry is very cost-sensitive. Our research indicates that the current trend of using large built-in caches is unnecessary and a waste of money and power for most users. Disk manufacturers could use much smaller built-in caches to reduce the cost as well as power-consumption, without affecting performance.

1. Introduction

Almost all modern hard disk drives have an internal buffer, or *built-in cache*. The disk built-in cache can serve multiple purposes. Besides as a pass-through speed-matching buffer between the bus and the disk media, it can be readily extended to include *read caching*, *readahead* and *write caching* [12, 10, 11].

In the last couple of years, hard disk manufacturers have dramatically increased the size of disk built-in caches in their products. Even as recently as the late 1990s, 256 to 512 KB was common on disk drives, and it was not unusual to find only 512 KB cache on even some SCSI units (though many had from 1 MB to 4 MB). Today, a 2 MB built-in cache is common on retail low-end IDE/ATA drives, and some SCSI drives are now available with 16 MB. There are possibly two main reasons for this dramatic increase in built-in cache sizes. The first is that memory prices have dropped precipitously over the last few years. The second is pertinent to marketing: hard disk consumers have a perception that doubling or quadrupling the size of the built-in cache will have a great impact on the system performance.

1.1. Do large built-in caches improve system performance?

While large built-in caches may show their usefulness under some artificial disk benchmarks, there are no published research results, as far as we know, to demonstrate that they can really improve the performance of real systems. In fact, we believe that large disk built-in caches will not significantly benefit the overall system performance. The reason is that *all modern operating systems already use large I/O buffer caches to cache reads and writes*. At the time of this writing, most low-end personal computers have at least 256 MB of RAM, and high-end workstations and servers have many gigabytes of RAM. As a result, the OS can use an I/O buffer cache ranging from tens of MBs to several GBs. With such a large OS buffer cache, the effectiveness of disk built-in cache is unclear. Consequently, it is quite meaningful to study the behavior of the built-in cache and the effect of this cache on the overall system performance. Such a study should use real-world workloads, with real system configurations. This paper uses detailed file system and disk system simulation to explore the effect of the built-in cache on the system performance, especially when the file system buffer cache gets larger and larger.

This research will benefit the disk drive industry and user community in the following two important ways. First, the

Proceedings of the 11TH IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS'03) 1526-7539/03 $ 17.00 © 2003 IEEE

disk drive industry is very cost sensitive and has a thin profit margin. A difference of merely several dollars may determine the success or failure of a product. If we can determine a smaller, optimal built-in cache size, the cost of disk drives can be reduced. Second, a large built-in cache uses more power. As many systems increasingly become power-aware, it is important to reduce the power consumption of disk caches. Undoubtedly, a smaller disk built-in cache will use less power.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 describes the simulation model. Section 4 introduces the analytic model. Section 5 shows the results of simulation and analyses of experiments. Finally, we summarize our work in Section 6.

2. Related Work

There are few studies conducted on the behavior of the disk built-in cache as the size of the file system buffer cache is becoming larger. Much work has been done in disk modeling. Ruemmler and Wilkes [12] present a strong case for detailed disk drive simulators and Ganger et al. have made available a highly-configurable disk simulator called DiskSim [3]. Shriver et al. [15] also present analytic models for modern disk drives. They all believe that the disk built-in cache needs to be modeled for caching if we want to get results closely matching the real disk drive. Moreover, some researchers have provided a suite of extraction techniques to obtain detailed knowledge of disk characteristics, including the built-in cache segment number, type and size [13, 19].

Ruemmler and Wilkes [11] have studied access patterns at the disk level using a simulator. They showed that readahead and NVRAM (Non-Volatile RAM) write caching at the disk can yield significant improvement in performance. Their systems had small system RAM sizes (32–96 MB) by today's standard. Shriver et al. [14] have constructed an analytic model for file system reads, trying to explain why file system prefetching works. They believe that the disk built-in cache plays an important role in prefetching. Biswas et al. [1] have also used a trace-driven simulation to explore the issues around managing a non-volatile write disk caches, and found that having a relatively small amount of NVRAM for a write cache is very effective in reducing the number of I/O writes to the disk and still allows for a write-behind strategy for the write cache to ensure consistency and crash recovery. Smith [16] has given design considerations (i.e., cache location, cache size, etc.) to both disk caches located in the I/O system and those located in the main memory.

Recent work by Wong et al. [18] have explored the benefits of *exclusive* caches, in which data blocks are either in the client or disk array cache, but never in both. Their work is motivated by a fact that modern disk arrays generally have a very big cache, typically in the size of several GBs. The disk built-in cache in our study instead is much smaller (typically several MBs).

3. Simulation Model

In order to conduct this research, we used various file system traces to drive a system model that uses a realistically-sized file system buffer cache and a modern disk drive with a large built-in cache. Our main performance metric is *file system response time*, which is the response time measured after a file read or write request is issued by an application.

3.1. Simulators

We developed a detailed file system simulator, which models the 4.4BSD Unix implementation of the Berkeley Fast File System [6, 17]. FFS and its variations are the most widely used file system for Unix systems. The algorithms of the simulator are ported from the BSD kernel source code. The file system simulator is built on top of a popular and accurate disk simulator called **DiskSim** [3] (due to space constraints, we here omit the detail of our simulators. Please refer to [20] for more detail). Based on our understanding of the FFS, we determined a set of parameters that allow us to model the FFS, which can be found in Table 1.

We chose the disk Quantum Atlas 10K (TM09100W) as a disk model in our experiments, because it has been already validated by DiskSim [2]. The disk capacity is 9GB with a built-in cache of 2 MB. Other parameters can be found in [2].

3.2. File System Workloads

In order to provide a fair and unbiased performance evaluation, we paid special attention in selecting the workload that drives our simulator since it plays a critical role in performance evaluation. Our main objective in choosing the workload is to use the real-world file system traces. In addition, a file system trace generated from PostMark [5] benchmark is used. We have also generated a small number of synthetic micro-benchmark workloads, which are used to study a specific aspect of system performance and will be discussed later.

3.2.1. Real-World Traces

Five real-world traces have been used in our simulation. **INS**, **RES**, and **WEB** [8, 9] are from University of California. **INS** is a collection of traces from a group of 20 machines located in labs for undergraduate classes. **RES**

parameter	definition	value
BlockSize	the amount of data which the file system processes at once	8 KB
FragmentSize	the minimum amount of data which the file system processes at once	1 KB
DirectBlocks	the number of blocks that can be accessed before the indirect block needs to be accessed	12
ClusterSize	the amount of files that is stored contiguously on disk	64 KB
CylinderGroupSize	the size of a cylinder group	16 MB
SysCallOverhead	time needed to check the file system buffer cache for the requested data	0.01 ms
MemoryCopyRate	rate at which data are copied from the file system buffer cache to the application memory	0.01 ms/KB

Table 1. Parameters for FFS simulator

traces	RES	INS	WEB	HARP	SITAR
data read(MB)	52743	94619	327838	520	213
data write(MB)	14105	16804	960	249	183
read:write ratio	3.7	5.6	341.5	2.08	1.16
reads(thousands)	9433	71869	9545	26	57
writes(thousands)	2216	4650	779	12	49
file size(0–16 KB)	60%	88%	63%	73%	80%

Table 2. Five real-world traces.

consists of 13 machines on the desktops of graduate students, faculty, and administrative staff of the research group project. **WEB** is collected from a single machine that is the web server for an online library project, which maintains a database of images using the Postgres database management system and exports the images via its web interface. This server receives approximately 2300 accesses per day during the period of the trace.

HARP and **SITAR** [4] are from University of Kentucky. **HARP** is gathered on a SPARC station for a research project. It is dominated by two collaborating programmers working on a large software project. Their work consists almost entirely of edit/compile/run/debug cycles on a large multimedia application. **SITAR** records user activity on a publicly available SPARC station. Most users accessed this workstation remotely via telnet, rlogin,and X-applications. It is a collection of file accesses by graduate students and professors doing work such as emailing, reading news, compiling programs, running LaTeX, editing files, and so on. Further detail of of these real-world traces can be found in [8, 9, 4]. Some basic characteristics of these real-world traces are described in Table 2.

3.2.2. PostMark Trace

The PostMark benchmark was designed to model the workload seen by Internet Service Providers under heavy load [5]. Specifically, the workload is meant to model a combination of electronic mail, netnews, and web-based commerce transactions. The PostMark trace is generated using 20,000 initially created files with a size range of 512 bytes to 16 KB. 200,000 transactions were performed with approximately 120,000 created and deleted files. There was no bias toward any transaction type. A total data size of approximately 1.1 GB is read and 1.2 GB is written.

4. Analytic Model

In this section, we present our analytic model which will be used to explain the simulation result in Section 5. We assume a uniprocessor running a modern file system (FFS). The system includes a large file system buffer cache, which holds disk blocks that have been referenced previously and uses an LRU replacement policy. To simplify our analysis, we assume that an I/O request issued by an application can be satisfied with a single disk access.

Here, we present several parameters that will be used in our model.

4.1. File System Specification

- $FSRT$ is the mean file system response time.
- T_{hit} is the amount of time that it takes to read the block from the file system buffer cache.
- T_{miss} is the amount of time that it takes to read the block from the disk.

- P_{miss} is the miss rate of the file system buffer cache.
- $RequestSize$ is the mean request size at the system level.
- Other parameters can be found in Table 1.

4.2. Disk Drive Specification

- DRT is the mean disk response time.
- $DiskOverhead$ includes the time to send the request down the bus and the processing time at the controller, which is made up of the time required for the controller to parse the request, check the disk built-in cache for the data, and so on [14].
- T_{seek} is the mean seek time.
- T_{rotat} is the mean rotational latency.
- T_{mech} is the service time of a request in the disk mechanism.
- $DiskRequestSize$ is the mean disk request size.
- $P_{DiskMiss}$ is the disk built-in cache miss rate.
- $BusTR$ is the data transfer rate between the disk built-in cache and the host. In our simulation, the transfer rate is 18 MB/s.
- $DiskTR$ is the data transfer rate between the disk surface and the disk built-in cache. The disk we used in our simulation spins at 10,025 RPM, thus giving us a $DiskTR$ of close to 18 MB/s.

4.3. Disk Response Time

The disk response time DRT depends both on the disk mechanism service time T_{mech} and the built-in cache miss and hit probabilities. In particular, $P_{DiskMiss}$ is the probability that a request will be a cache miss. If readahead at the disk is not enabled or if a request is a write (since the disk built-in cache is made of volatile RAM, it by default uses write-through to preserve data in case of a power failure), $P_{DiskMiss}$ will be 1 [15].

T_{mech} depends on the disk seek time, the rotational latency and the transfer time. It can be computed as follows:

$$T_{mech} = T_{seek} + T_{rotat} + \frac{DiskRequestSize}{DiskTR} \quad (1)$$

So the disk response time DRT can be computed as:

$$DRT = DiskOverhead + \frac{DiskRequestSize}{BusTR} + P_{DiskMiss} \times T_{mech} \quad (2)$$

4.4. File System Response Time

In our model, an access that hits in the file system buffer cache experiences time T_{hit} to read the block from the cache.

$$T_{hit} = SysCallOverhead + \frac{RequestSize}{MemoryCopyRate}$$

In the case of a cache miss, the block needs to be fetched from disk before it may be delivered to the application.

$$T_{miss} = T_{hit} + DRT$$

Given a cache miss rate P_{miss}, the file system response time can be computed as:

$$FSRT = T_{hit} + P_{miss} \times DRT \quad (3)$$

Combining equation (2) and (3), we have

$$FSRT = T_{hit} + P_{miss} \times (DiskOverhead + \frac{DiskRequestSize}{BusTR} + P_{DiskMiss} \times T_{mech}) \quad (4)$$

5. Simulation Results and Performance Analyses

This section presents our simulation results and performance analyses. We try to investigate the impact of disk built-in caches on the file system response time in the following aspects: cache size, readahead and write caching. Due to space constraints, we cannot present all results here. Please refer to [20] for detailed results.

5.1. Impacts of Sizes of File System Caches and Disk Built-in Caches

Figure 1 shows the file system response time with different disk built-in cache sizes and file system buffer cache sizes for five real-world workloads as well as PostMark trace. The figure clearly shows that, with a file system buffer cache size of 16 MB or more, *the built-in cache has very little impact on the file system response time when the disk built-in cache size is larger than 512 KB*. When the file system buffer cache is 16 MB, a system using a 16 MB disk built-in cache is only 1.1–4.1% faster than a system with a 512 KB disk built-in cache. When the file system buffer cache is 128 MB, the former is only 0.3–1.5% faster than the latter. We therefore conclude that, larger built-in caches have virtually no noticeable performance benefit on overall system performance, especially when the file system buffer cache is large.

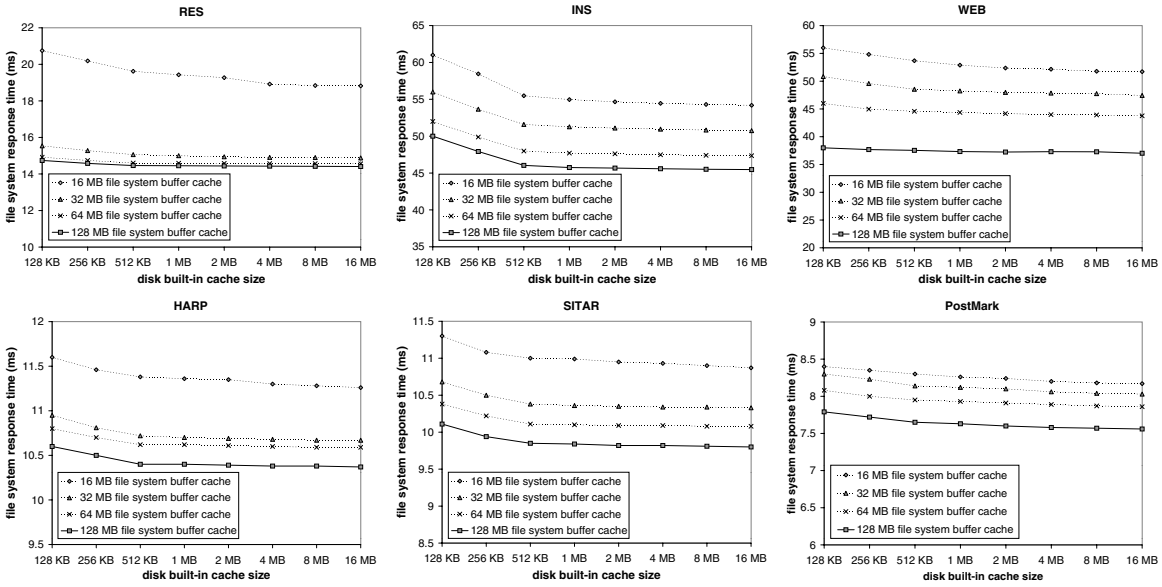


Figure 1. File system response time with different cache sizes for six workloads. Readahead at the disk level is enabled. Write caching at the disk level is disabled.

5.1.1. Analyses

In this section we use the analytic model discussed earlier to explain why this happened. For a certain workload with the same file system buffer cache size, it is reasonable to assume that T_{hit}, P_{miss}, $DiskRequestSize$, $DiskOverhead$, $BusTR$ and T_{mech} are same. So when the disk built-in cache size changes, we compute $\Delta FSRT$ from equation (4) as

$$\Delta FSRT = P_{miss} \times \Delta P_{DiskMiss} \times T_{mech} \qquad (5)$$

With the disk built-in cache increasing from 512 KB to 16 MB, $\Delta P_{DiskMiss}$ is quite small. This is because that, compared to the size of the file system buffer cache, the disk built-in cache is typically small, which means that most accesses never reach the disk. In addition, the disk built-in cache acts as a speed-matching circular buffer, "random" and "reuse" cache hits are relatively rare. Though requests for contiguous extension of prior reads can be common and performing readahead at the disk level can improve cache hit rate, the cache hit rate as a result of readahead is still low because of multiple interleaving streams of read requests. So even the disk built-in cache reaches up to 16 MB, $P_{DiskMiss}$ is still quite large. In our simulation, $P_{DiskMiss}$ is always larger than 90%. As a result, $\Delta P_{DiskMiss}$ is still small ($< 10\%$) as the built-in cache varies from 512 KB up to 16 MB. Furthermore, when the file system buffer cache becomes larger, P_{miss} decreases. For example, when file system buffer cache is 32 MB, P_{miss} is less than 15% for all workloads (except for WEB and PostMark, their P_{miss} are about 45% and 96% respectively). According to equation (5), $\Delta FSRT$ is small consequently.

We also found that there is a slight difference in the mean physical access time as the file system buffer cache size varies. This suggests that the "filtering" of the reference stream by various file system buffer caches alters the interaction between requests presented to the disk, resulting in the difference in the mean physical access time.

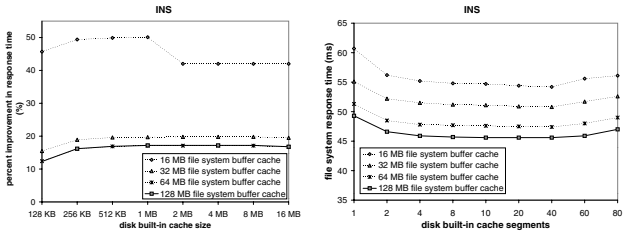
Moreover, the disk built-in cache miss rate increases slightly when the file system buffer cache increases from 16 MB to 128 MB. This is because that larger file system buffer caches capture more locality, those requests missing the file system buffer cache therefore have poorer locality, resulting in higher built-in cache miss rates.

5.2. Impact of the Readahead Policy

To improve the disk built-in cache hit rate for sequential access patterns, modern disk drives prefetch data after read requests. If a next read comes, the disk will check if the requested data is in the cache as a result of readahead for previous requests. On a cache hit, the disk sends the data directly from the disk built-in cache, and thus avoids mechanical positioning overheads (seek time plus rotational latency). In the disk model we used, 9 out of 10 cache segments are devoted for the readahead cache.

We experimented on the effect of disabling or enabling readahead for all the traces (due to space constraints, we here do not present the figure). The percent improvement of enabling readahead in response time for PostMark trace is very small (about 1.8 – 1.9%), this is because the incidence of each transaction type and its affected files in PostMark are chosen randomly, thus minimizing the influence of file system caching and disk level caching [5]. Even though all files are read in their entirety, the small size of the randomly chosen files (ranging from 512 bytes to 16 KB, which can be read in only one disk request) keeps readahead from performing effectively. While for RES, INS, WEB, HARP and SITAR, when the file system buffer cache is 16 MB, the percent improvements of enabling readahead in response time are 13.5%, 42.1%, 6.0%, 9.0% and 10.5% respectively. As the file system buffer cache size increases, the percent improvements decrease dramatically. This is because larger file system buffer caches mean higher cache hit, and can absorb most reads and writes, thus amortizing the negative effect of not performing readahead to some extent. There is an exception for INS: when the file system buffer cache varies from 16 MB to 128 MB, the percent improvements in response time are more than 17.7% with performing readahead. This is because INS has much more reads than writes and it also has a great number of small files (88% for file size less than 16 KB) [9, 8]. However, the percent improvements for other four real-world traces are small.

5.2.1. Effects of Cache Sizes



(a) (b)

Figure 2. Performance impact on INS. Write caching at the disk is disabled. (a) Percent improvement of enabling readahead in response time with different file system buffer cache sizes and disk built-in cache sizes. (b) Performance impact of numbers of cache segments.

Readahead so far was studied by using a 2 MB disk built-in cache. In an attempt to further explore the effect of readahead under various cache sizes, we ran our simulation for five real-world workloads under various file system buffer cache sizes and disk built-in cache sizes with readahead enabled and disabled (PostMark was excluded in this experiment since readahead has a very little impact on it). Figure 2(a) shows the results for INS trace. We observed similar results for other four real-world traces as well.

Clearly, the size of file system buffer caches affects the effectiveness of performing readahead at the disk level. When the file system buffer cache size is 16 MB, readahead performs extremely well and achieves about 42–45% improvement in response time for various disk built-in cache sizes. While the file system buffer cache size increases (up to 32 MB, 64 MB and 128 MB), the percent improvement (about 12–20%) decreases dramatically. This suggests that larger file system buffer caches mean higher cache hit and can absorb a greater fraction of the read requests, therefore amortizing the negative effect of not performing readahead at the disk to some extent. Furthermore, we observe that, for various file system buffer cache sizes, a 512 KB disk built-in cache performing readahead nearly achieves a maximum percent improvement in response time over one without performing readahead. This makes us to believe that a 512 KB built-in cache can perform readahead very well, even compared to a larger one.

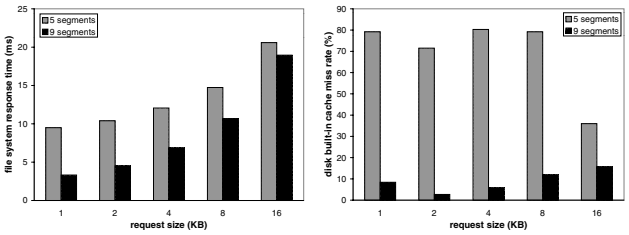
5.2.2. Effects of Numbers of Cache Segments

A single readahead cache can provide effective support for only a single sequential read stream. If two or more sequential read streams are interleaved, the result is no benefit at all. This can be remedied by segmenting the cache so that several unrelated data items can be cached.

Segmenting a disk cache is similar to dividing a CPU cache into multiple cache lines. Just like in the CPU cache, there is a tradeoff between the number of segments and the segment size. More segments in the cache can accommodate a larger number of concurrent workloads. On the other hand, when the cache size is fixed, having more segments means a smaller segment size. This will have a negative impact on the cache's ability to absorb spatial locality, because the readahead algorithm will have less room to put the prefetched data.

The disk model we used has a default segment number of 10. To study the impact of segment numbers, we varied the number of segments from 1 to 80 while keeping the built-in cache size fixed at 2 MB. The results for INS trace are shown in Figure 2(b). We observed similar results for other cache sizes. Note that, if the number of segments is small (1–8), the cache cannot accommodate many concurrent requests in the workload, rendering readahead useless. For this particular trace, the number of concurrent access streams is probably less than 10. As a result, there is vir-

tually no performance difference when the number of segments varies from 10 to 40. Further increasing the number of segments will decrease performance instead because each segment becomes too small. For example, using 80 segments resulted in a performance decrease of about 2.5%.



(a) (b)

Figure 3. The file system buffer cache is 16 MB and the disk built-in cache is 2 MB. Write caching at the disk is disabled. (a)File system response time for sequentially reading nine 64 KB files with 5 readahead cache segments and 9 readahead cache segments. (b)Disk built-in cache miss rate for both 5 readahead cache segments and 9 readahead cache segments.

In an attempt to further quantify the effect of readahead, which depends on a number of factors, including the number of cache segments, the number of concurrent sequential workloads, and request sizes, we generated a set of synthetic micro-benchmark traces for our simulation. This set of traces consists of 9 concurrent workloads sequentially reading 64 KB files with various request sizes from 1 KB to 16 KB.

Figure 3(a) compares the file system response time when servicing 9 concurrent workloads with 5 cache segments and the response time with 9 cache segments. When the request size takes 1 KB, 2 KB, 4 KB, 8 KB and 16 KB, the decreases in the response time are 65.1%, 56.6%, 43.2%, 27.5% and 7.9% respectively. Obviously, 9 cache segments perform much better than 5 segments in concurrently reading nine files. This is because 9 cache segments can prevent data prefetched into cache segments from being polluted by other read requests, and consequently improve the cache hit rate for sequential access patterns. Figure 3(b) compares the disk built-in cache miss rate for both 5 segments and 9 segments with various request sizes. We can easily see that, there is a big difference in disk built-in cache miss rates between 5 segments and 9 segments. Therefore, if there are enough cache segments to service multiple streams of sequential requests, readahead will work more effectively.

In other words, if the file system can dynamically adapt the number of the disk built-in cache segments to be the number of files being concurrently read from the disk, the built-in cache will perform better. This is a simple and inexpensive SCSI operation, and can make the built-in cache work effectively. With the request size increasing, however, the benefit will decrease. A smaller request size can yield more benefits in readahead when the number of the cache segments is equal or larger than the number of concurrent workloads.

5.3. Impact of Write Caching at the Disk

Enabling write caching at the disk could improve the performance in the following ways. First, write latency can be reduced due to the fact that write requests are considered completion as soon as all write data have been transferred to the cache (but not necessarily to disk media). Second, data in a write cache segment are often overwritten in place, reducing the amount of data that must be written to the disk media. Third, stored writes at the write cache segment make it possible for the controller to schedule them in near-optimal fashion, so that each write takes less time to perform. In the disk model we used, one cache segment ($\approx$ 0.2 MB) was devoted for write caching. This is because: (1) More than one cache segments devoted for write caching have little positive impact on overall performance in our experiments, and (2) as discussed earlier, more cache segments devoted for readahead can be more effective.

Our results show that there is almost no file system response time improvement in WEB when write caching is enabled. Since the WEB workload has few writes, its disk traffic is dominated by reads even with larger file system buffer caches [9, 8]. While for RES, INS, HARP, SITAR and PostMark, the improvements in file system response time are 3.0–5.8%, 2.0–3.4%, 4.1–6.0% , 5.0–7.0% and 35.6–37.8%, respectively (due to space constraints, we here do not present the figure). With the file system buffer cache increasing from 16 MB to 128 MB, we found that the percent improvements increase. This is because these workloads like RES and INS have more write traffic while the file system buffer cache increases [8, 9]. In other words, larger file system buffer caches can alter the workload presented to the disk by absorbing a greater fraction of the read requests [7]. Since most write requests must eventually be reflected on the disk for safety, disk traffic and disk performance will become more and more dominated by writes. Our results suggest that, in write-intensive workloads, write caching can provide much more benefits. We also proposed for the built-in cache a new policy — *selective write caching*, wherein metadata requests and user-data requests are treated differently at the disk level (please refer to [20] for more detail). Our results show that selective

write caching performs better than no write caching (except for WEB workload) without compromising system consistency.

6. Conclusions

Disk drive manufacturers are putting increasingly larger built-in caches into disk drives. Some disk drives nowadays have a cache of 16 MB or bigger. However, there are few published results to demonstrate that such large built-in caches can really improve overall system performance. While most of the research work has been busy trying to come up with better caching schemes and finding new ways to use caches, our work takes the opposite view. Via detailed simulations, we arrive at the following three main conclusions:

(1) *Given the workloads studied, larger disk built-in caches do not have much impact on system performance*. Since the disk drive industry is very cost-sensitive, we argue that disk manufacturers could use much smaller disk built-in caches (about 512 KB) in future products to bring down the cost, without affecting performance. Moreover, a smaller cache will consume less power, which is becoming increasingly important for many portable applications.

(2) Enabling readahead at the disk provides significant performance improvements for workloads with read sequentiality (e.g., multimedia applications might be the case), *but if the number of the disk built-in cache segments is smaller than the number of concurrent workloads, performing readahead at the disk has little effect on file system response time*. Therefore, the number of built-in cache segments is important for multiple sequential streams of read requests when the disk built-in cache performs readahead.

(3) Write caching at the disk also has some positive effects on system performance (especially on write-intensive workloads), at the risk of data loss and the need for more complicated error recovery after a power failure. We propose selective write caching to strike the balance between the performance and the risk of file system inconsistency.

References

[1] P. Biswas, K. K. Ramakrishnan, and D. Towsley. Trace driven analysis of write caching policies for disks. In *Proceedings of the 1993 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 13–23, Santa Clara, CA, May 1993.

[2] G. R. Ganger and J. Schindler. Database of validated disk parameters for disksim. http://www.ece.cmu.edu/-ganger/disksim/diskspecs.html., Feb. 2000.

[3] G. R. Ganger, B. L. Worthington, and Y. N. Patt. The disksim simulation environment version 2.0 reference manual. http://citeseer.nj.nec.com/333878.html, Dec. 1999.

[4] J. Griffioen and R. Appleton. The design, implementation, and evaluation of a predictive caching file system. Technical Report CS-264-96, Department of Computer Science, University of Kentucky, June 1996.

[5] J. Katcher. Postmark: A new file system benchmark. Technical Report TR3022, Network Appliance Inc., Oct. 1997.

[6] M. K. McKusick, K. Bostic, M. J. Karels, and J. S. Quarterman. *The Design and Implementation of the 4.4BSD Operating System*, chapter 8, pages 285–300. Addison-Wesley Publishing Company, first edition, 1996.

[7] J. K. Ousterhout, H. D. Costa, D. Harrison, J. A. Kunze, M. Kupfer, and J. G. Thompson. A trace-driven analysis of the unix 4.2BSD file system. In *Proceedings of the 10th ACM Symposium on Operating Systems Principals*, pages 15–24, Orcas Island, WA, Dec. 1985.

[8] D. Roselli. Characteristics of file system workloads. Technical Report CSD-98-1029, Computer Science Division, University of California, Berkeley, Dec. 1998.

[9] D. Roselli, J. R. Lorch, and T. E. Anderson. A comparison of file system workloads. In *Proceedings of the 2000 Annual Technical USENIX Conference(USENIX-00)*, pages 41–54, San Diego, CA, June 2000.

[10] C. Ruemmler and J. Wilkes. Modeling disks. Technical Report HPL-93-68 revision 1, Hewlett-Packard Laboratories, July 1993.

[11] C. Ruemmler and J. Wilkes. UNIX disk access patterns. In *Proceedings of Winter 1993 USENIX*, pages 405–420, San Diego, CA, Jan. 1993.

[12] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–28, 1994.

[13] J. Schindler and G. R. Ganger. Automated disk drive characterization. Technical Report CMU-CS-99-176, Carnegie Mellon University, Dec. 1999.

[14] E. Shriver, C. Small, and K. A. Smith. Why does file system prefetching work? In *Proceedings of the 1999 USENIX Annual Technical Conference*, pages 71–83, Monterey, CA, June 1999.

[15] E. A. M. Shriver, A. Merchant, and J. Wilkes. An analytic behavior model for disk drives with readahead caches and request reordering. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems(Sigmetrics'98/Performance'98)*, pages 182–191, Madison, WI, June 1998.

[16] A. J. Smith. Disk cache-miss ratio analysis and design considerations. *ACM Transactions on Computer Systems*, 3(3):161–203, Aug. 1985.

[17] U. Vahalia. *UNIX Internals — The New Frontiers*. Prentice Hall, 1996.

[18] T. M. Wong and J. Wilkes. My cache or yours? making storage more exclusive. In *Proceedings of the USENIX Annual Technical Conference*, pages 161–175, 2002.

[19] B. L. Worthington, G. R. Ganger, and Y. N. Patt. On-line extraction of scsi disk drive parameters. Technical Report CSE-TR-323-96, Department of Electrical Engineering and Computer Science, University of Michigan, Dec. 1996.

[20] Y. Zhu and Y. Hu. Can large disk built-in caches really improve system performance. Technical Report TR259/03/02ECECS, University of Cincinnati, Mar. 2002.