Measuring Parameters of the EXT2 File System

Dan Gibson
University of Wisconsin-Madison Department of Electrical and Computer Engineering
degibson@wisc.edu

Abstract

File system performance directly impacts performance of modern computer systems. It is important to design programs with this impact in mind. To accomplish this, many file system attributes must be measured or determined by other means.

In this project, files system parameters were measured for the EXT2 file system. Disk block size was determined to be 4096 bytes. Prefetch buffer size was measured as 128k-bytes. The size of the file cache is two megabytes. Measurement also reveals that each inode has exactly twelve pointers to direct data blocks.

1 Introduction

Performance of the file system can have a dramatic effect on the performance of disk I/O-dominated programs. Awareness of file system sizings can improve the runtimes of programs which optimize for the parameters of the underlying file system. The disk block size, prefetching size, file cache size, and the number of direct pointers in a given inode were measured for the EXT2 file system [1] with empirical experiments employing high-resolution counters to measure I/O blocking times.

It was determined that for the particular configuration of EXT2 used in our experiments, the disk block size is 4096 bytes. The file system prefetches large files on 128k-byte boundaries. The file cache is approximately 2MB in size, and there are twelve direct pointers per inode.

The primary goal of this project was not to measure file system attributes, but to experience measurements of computer systems, especially the difficulties inherent in designing experiments and interpreting empirical data.

The remainder of this paper is organized as follows: Section 2 discusses the methodology used by the author; Section 3 presents empirical results; Section 4 outlines conclusions drawn from the data.

2 Methodology

High-resolution counters were used extensively in this project, provided by the Intel ® x86 instruction `rdtsc`. This instruction provides a 64-bit cycle-accurate count of elapsed clock cycles since the counter was last reset. Extensive timer validation was performed using more accessible but less accurate system calls, namely `gettimeofday`() and `sleep`(). A discussion of the timer validation study follows in Section 4, however, the exact number of cycles corresponding to a unit of time was not determined (nor was it required). It is sufficient to validate that the value returned by `rdtsc` instruction increases linearly in time—thereby changes in the value returned by `rdtsc` correspond linearly to actual time elapsed, and provides reliable comparative values of elapsed time.

The primary means by which measurements were taken of the EXT2 file system was by measuring elapsed time of system calls `read`(), `write`(), and `fsync`(). For the measurement of disk block size, a large number of small, sequential reads (8 bytes in size) were performed; disk block boundaries produced periodic steps in the observed total latency of all `read`() system calls.

The disk prefetching size was measured with a sequence of larger reads (512 bytes in size), with a significant delay (1-10 ms) inserted between reads, to provide sufficient time for prefetching. Periodic increases in latencies, similar to those discovered when measuring disk block size (but much larger in period), expose the prefetching mechanism's buffer size.

The file cache size experiments required more sophisticated techniques. A number of blocks (the current working set) was accessed in reversed order, measuring the average `read()` latency. The size of the working set was gradually increased over time. Between changes in the working set size, the file cache was flushed by unmounting and re-mounting the entire file system. Before each measurement, the cache was "warmed-up" with a `read()` to each block in the working set, to eliminate the impact of compulsory misses. In the absence of other file-I/O activity, the remaining cache misses are due to capacity misses, assuming the caching mechanism does not cause conflict misses.

The number of direct pointers per inode was measured with a series of sequential, block-sized writes to a new file. A consistent spike in latency of `write()` and `fsync()` after a fixed number of writes betrays the allocation of the file's first indirect block (requiring two writes—the indirect block and a data block, instead of a data block alone). Assuming the file is written in such a manner as to use the direct block pointers first, the allocation of the indirect block occurs immediately after the last direct block has been filled.

All experiments were performed on a machine running Linux 2.4.26 (Slackware 10 distribution [2]) and the EXT2 file system [1]. In all experiments, background processes (approximately 20) accounted for less than 0.01% of all execution time. The underlying hardware consists of an Intel ® Celeron ® processor, clock speed 1.70 GHz, with 128k-byte on-chip caches. Main memory is 64 Mb in size. The disk used for the experiments is approximately 10 Gb in size, accessed over a 133 Mhz ATA bus.

3 Results

The experiments described in Section 2 rely heavily on the accuracy (linearity) of high-resolution timers. The accuracy of the high-resolution counter was examined for times much longer than needed for the experiments above to measure linearity as compared to true elapsed time, given by `gettimeofday()`. Figure 1 displays the results of the timer validation. The value returned by the high-resolution timer is overlayed with the function depicted in Figure 1. The deviation is extremely small, and the observed slope corresponds closely to the expected value of 1.70 Ghz.

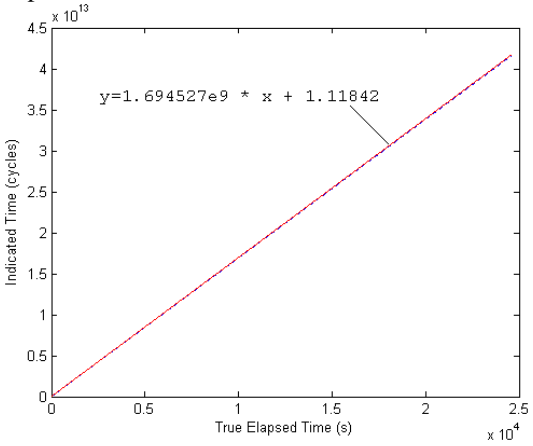

Figure 1 – Indicated Time versus True Elapsed Time

Measurement of disk block size called for a large number of small, sequential reads (8 bytes in size). Disk block boundaries produced periodic steps in the observed total latency of all `read()` system calls. Figure 2 depicts this periodic step, occurring every 4096 bytes. This value correlates strongly against the value selected for disk block size on installation of EXT2 (4k-byte).

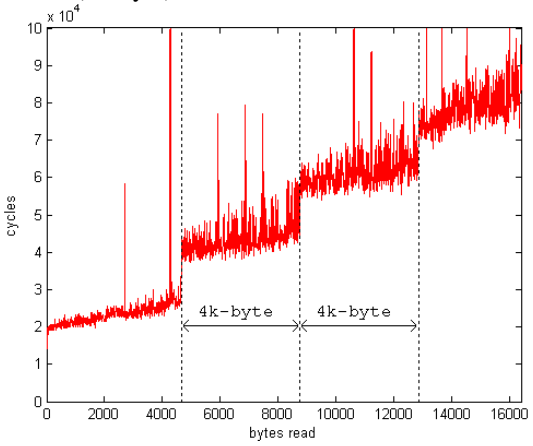

Figure 2 – Total latency versus bytes read

Significant noise is observable in Figure 2 and subsequent figures. Small amounts of noise are attributable to variations in out-of-order execution times—large spikes are attributable to unexpected interrupts and/or context switches.

The disk prefetching size was measured with a sequence of larger reads (512 bytes in size), with a significant delay (1-10 ms) between reads, to provide sufficient time for prefetching. Periodic increases in latencies, similar to those discovered when measuring disk block size (but

much larger in period), expose the prefetching mechanism's buffer size. Figure 3 depicts these periodic latency spikes.

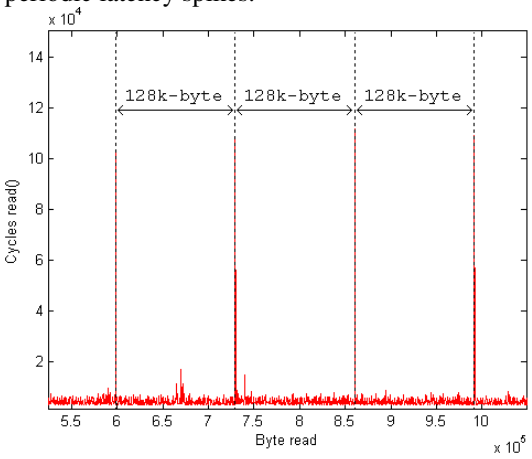

Figure 3 – Cycles per `read`() versus byte request

These spikes are of significant size compared to typical `read`() latencies, accounting for disk accesses. They are aligned precisely on 128k-byte boundaries. This strongly indicates a prefetch size of 128k-bytes.

Measurement of the file cache size called for a varying number of blocks (the current working set) to be accessed, measuring the average `read`() latency. The size of the working set was gradually increased over time. By measuring average `read`() latency, one can determine when blocks begin to experience cache-misses, characterized by a step in average latency. This step is observable in Figure 4.

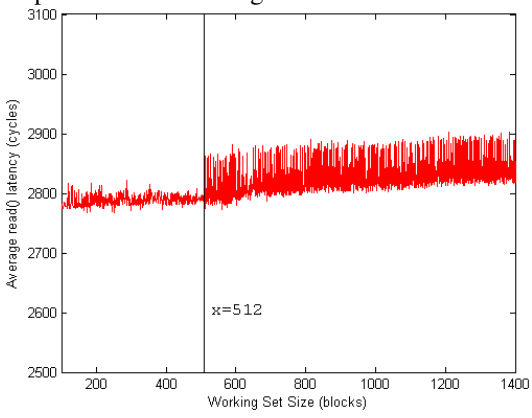

Figure 4 - Average read() latency versus WS Size

The step observable in Figure 4 occurs at approximately 512 blocks, or 2MB. This strongly suggests an observed file cache size of 2MB.

The number of direct pointers per inode was measured with a series of sequential, block-sized writes to a new file. A consistent spike in latency of `write`() and `fsync`() after a fixed number of writes betrays the allocation of the file's first indirect block (requiring two writes—the indirect block and a data block, instead of a data block alone). This spike is observable in Figure 5.

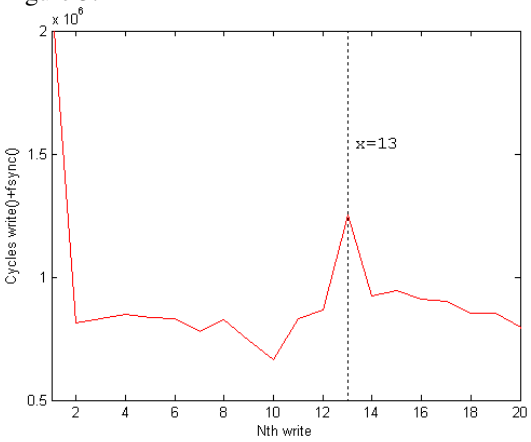

Figure 5 – Write latency versus blocks written

The 13th write has a latency approximately twice that of writes 12 and 14. This is due to the allocation of an indirect block on the 13th block-sized write, requiring an extra disk write. This implies there are precisely twelve direct pointers in a given file's inode. The large observed latency on the first write can be attributed to disk seek time.

4 Conclusions

High-resolution counters were used to measure parameters of the EXT2 file system. For a particular configuration, disk block size, prefetch size, file cache size, and the number of direct pointers per inode were measured with a high degree of confidence.

Attribute	Size
Disk Block Size	4096 bytes
Prefetch Size	128k-bytes
File Cache Size	2M-bytes
Direct ptr. Per inode	12

The above values were determine through timed observation of file system performance, and are based on empirical evidence, and validated in the case of block size by reference to EXT2 documentation.

As with many versions of Linux, this particular system uses all available free memory for file caching. While the background processes were not computationally intensive, the low amount of available memory on this machine resulted in a small file cache in main memory. Use of the Linux utility free reveals approximately 2-3 MB available memory (DRAM, not cache) in the system with these background processes. This accounts for the estimated file cache size of 2 MB.

The primary insights gained in this experiment are unrelated to the parameters actually measured: The difficulty in measuring computer systems with confidence should not be underestimated. A certain degree of care must be exhibited when designing experiments and interpreting their results. Without careful thought and restraint, it is very possible to measure the wrong phenomenon, or measure the correct phenomenon in an incorrect manner.

References

[1] Card, R., Ts'o, S., and Tweedie, S. "Design and Implementation of the Second Extended Filesystem." In Proceedings of the First Dutch International Symposium on Linux.

[2] http://www.slackware.com