Linux File Streaming

Ahmed Ayad
ahmed@cs.wisc.edu

Nermeen Bassiouny
nermeen@cs.wisc.edu

Abstract

In this paper, we are investigating a claim about the performance of the Linux ext2 file system under streaming I/O. A look on the ext2 does reads and writes is presented together with an extensive testing of the file system performance under streaming reads and writes. The tests aim at comparing the performance of ext2 with that of the raw performance of the underlying storage device.
The results of the benchmarks show no evidence of a problem in ext2 performance. Ext2 is able to exploit the full bandwidth of the underlying storage device in both reading and writing at its worst case; that is, without the help of its buffering techniques.

1. Introduction

Linux is a Unix-like operating system suitable for the PC environment. Linux has gained a worldwide acceptance since its first release in the early 90s. The fact that Linux is both free and open source was the major contribution towards its wide use and acceptance. This fact also makes Linux an excellent environment for research and experimenting. After all, this is how it was developed in the first place.
With the increasing use of multimedia files, file system support to streaming I/O is crucial to the performance of multimedia applications. The performance gain is generally achieved through clever caching techniques using file system buffers. Those techniques, however, are of limited effect if the file system is unable to fully utilize the full bandwidth of the underlying storage device. The purpose of this paper is to investigate the performance of the Linux ext2 file system under streaming I/O. One of the main goals is to verify a claim about the ext2 performance. The claim is that there exists a performance gap between the ext2 performance and the raw throughput of the underlying storage device. If the problem is verified, we should proceed with more investigation of the source of the problem and propose an appropriate solution.
To verify the problem, we designed a series of benchmarks to compare the ext2 performance with the raw device performance. The outcome of the benchmarks did not confirm the existence of a problem in the file system performance. Hence, this paper is solely dedicated to the description of the benchmarks and the discussion of the resulting numbers. The benchmarks were done on two different systems having different types of storage devices. So a side benefit of this paper is to serve as a benchmarking report on the performance of these two types of storage devices and the performance of ext2 on top of them.
The rest of the paper is organized as follows: section 2 looks at how the ext2 file system performs its read and write optimizations. Section 3 describes the setting on which the different benchmarks were run plus a detailed description of each experiment and its results. Finally, section 4 concludes the paper with the major findings.

2. The Linux Ext2 File System

The Ext2 provides standard Unix-like file system functionalities. It has the same notions of regular files, directories, special files and symbolic links. A detailed description of the design and features of ext2 can be found in [1]. In the next two subsections we will explain briefly the read and write process for the ext2 file systems and the optimizations done to improve their performance.

2.1. Ext2 Reads

The Ext2 file system uses the function `generic_file_read` (kernel function) to perform read operations. This function tries to determine if the file is being read sequentially or randomly. If the file is accessed sequentially it tries to read ahead the next pages in the file. It is important to understand the read ahead mechanism before explaining the read operation. There are two global constants MIN_READAHEAD (3 pages) and MAX_READAHEAD (33 pages). When a read operation is issued on a certain file, a number of pages in a previously set read ahead window – which starts initially at the MIN_READAHEAD value - are read. For the next read, if the read falls in the previously read window and in the second half of it, the read ahead limit is doubled but it is not allowed to go above MAX_READAHEAD. If the second read is in the read ahead window, but didn't cross the half window boundary, the read ahead window is kept at its size. Finally, if the read is outside the previously read window, it is deduced that the file is being read randomly and the read ahead is disabled.

That said, we can briefly explain the `generic_file_read`:

- If the current read is in the set read ahead window, then the file is accesses sequentially. If it is not then the file is accessed randomly and the read ahead is disabled.
- It calls `find_page` to see if page exists in cache.
- If page is not found, a page is allocated by calling `get_free_page`. The page is added to the cache.
- The required data is reads from disk into the allocated page.
- Whether the page was already in cache or was brought from disk to cache, the contents of that page are copied from the page to the buffer.

The performance of this read operation will be studied in detail.

2.2. Ext2 Writes

The write operation is implemented explicitly in the ext2 file system. Some optimizations were done in order to improve the write performance.

- Related inodes and data are clustered together in block groups. This reduces head seeks made when reading an inode and its data blocks.
- When writing data to a file, Ext2fs pre-allocates up to 8 adjacent blocks when allocating a new block. If the pre-allocated bock is not used it is returned back to the pool of free blocks. This pre-allocation achieves good write performances under heavy load. It also allows contiguous blocks to be allocated to files, which could speed up sequential file reads.

The write operation of the Ext2 file system proceeds as follows:

- The arguments are checked for errors in the write request. This includes the check for write flag, max size of ext2 files.
- If the file is opened with the Sync flag, the metadata of the file is also set to be synchronous.
- It performs a write loop as long as there is remaining data to be written.
- It allocates a buffer to write the data. It writes the data in the buffer and marks the buffer as dirty by calling the function `mark_buffer` dirty.
- Once all the data is written the file inode and descriptor are updated.

3. Benchmarks Description

3.1. Environment Settings and parameters

The benchmarks were done on two different environments. The first one is a PC with a Pentium III 667MHz processor with 128 MB of main memory and an 8GB IDE hard drive. The second is the Wisconsin DB cluster of workstations each with a dual Xeon 550MHz processor and a GB of main memory and an 8GB IBM SCSI hard drive. Henceforth, the first setting will be referred to as the IDE setting and the second will be referred to as the SCSI setting.
Some general notes on the parameters of the benchmarks:

1- The measured value in all the experiments is the data rate in mega bytes per second (MB/s). Data rate is measured by simply measuring the wall clock time of the operation and dividing it by the size of the data read or written.
2- File sizes in almost all the tests start at 50 MB to 1GB in SCSI and 50MB to 250MB in IDE, with a step size of 50 MB. The reason for the different end sizes in both settings is due to the fact that the SCSI machine had a 1GB of memory so we had to go up to 1GB of file size to preclude the effect of the file system buffer cache. On the other hand a 250MB file can easily overwhelm the 128MB memory of the IDE machine.

3.2. Read Benchmarks

3.2.1. Sequential read

This benchmark compares the performance of the Ext2 file system to the performance of the read from the raw device. A sequential read of incremental file sizes was performed. In this experiment, and all subsequent ones, we are reading portions of a big file on the drive. For every read size the data rate is measured in MB/sec. The read is performed in chunks of 1 MB each. The first read result for the ext2 was ignored in order to see the buffering effect. The trend of the performance is the same in the two systems. The raw disk rate is steady for all file sizes at 20MB/sec for the SCSI drive and about the same for the IDE drive. In both cases the read starts at a high rate due to the use of the buffer cache. When the memory is exhausted (depending on the memory size on each machine) the performance of the reads saturates at the same rate of the raw disk reads. Fig(1) and fig(2) show the result of this benchmark on the SCSI drive and the IDE drive respectively.

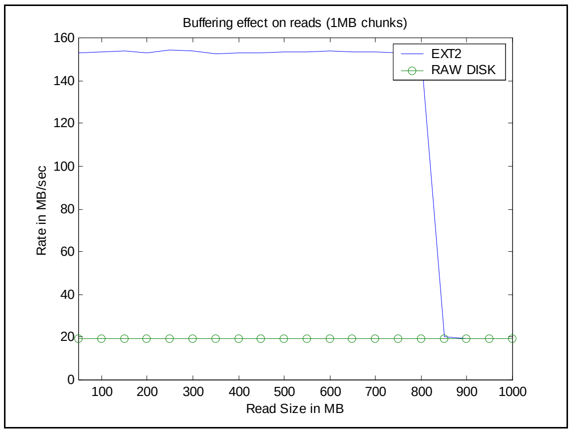

Figure 1. Sequential reads on SCSI

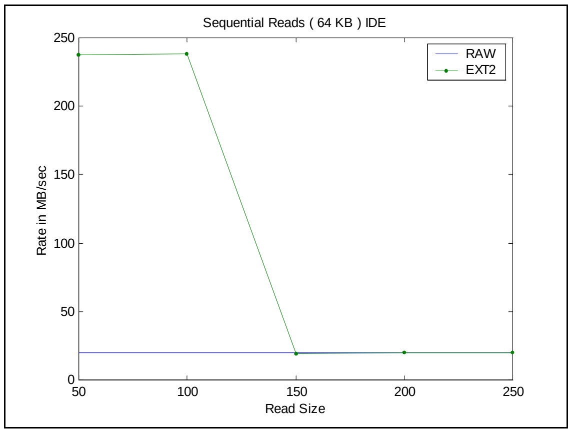

Figure 2. Sequential reads on IDE

3.2.2. Comparison with memory mapped I/O

This benchmark compares the performance of the Ext2 file system to the performance of memory-mapped I/O. The same settings of the previous experiment except that for the case of memory mapped I/O, instead of reading the chunk from the file in memory using the normal read interface, the read size was mapped into the program and then copied from the mapped area into the buffer. The raw disk performance was also measured in that case to compare the Ext2 and the mmap() interface to the raw bandwidth of the disk. The trend of the performance is the same. The raw disk rate is steady for all file sizes at 20MB/sec. The rate is high in the beginning for both the normal read and the mmapped read due to the use of the buffer cache. When the memory is exhausted (at around 700MB) the performance of the reads starts to saturate at the same rate of the raw disk reads.

It is important to note in this plot that the read rate in case of mmap() interface starts at a lower rate that the normal read interface. This can be attributed to the use of virtual memory in the mmap(). Even though the page might be in the file system's buffer cache, there is still the overhead of trapping into the virtual memory system. Fig(3) shows the result of this benchmark on the SCSI machine.

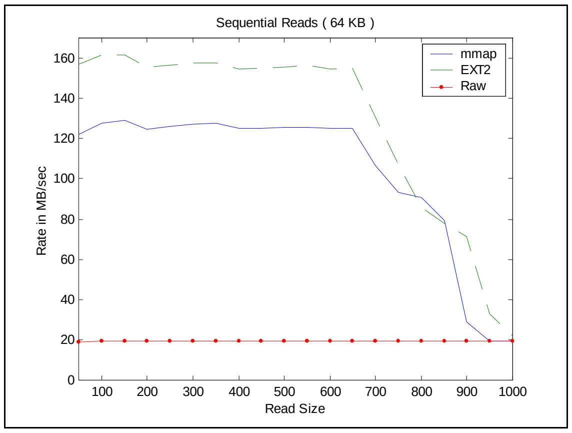

Figure 3. Sequential reads using memory-mapped I/O

3.2.3. Buffer size effect on read performance

To answer the question of whether the 1MB file in the previous benchmarks is significant with regard to performance, this benchmark measures the effect of the change of the buffer size on the read performance of the ext2 file system. This is also compared to the performance of the raw disk reads. A file of size 1G is read sequentially with incremental buffer sizes. Buffer sizes start at 4KB up to 16 MB and doubling each time. Again the same test is performed for the SCSI and the IDE drive. Fig (4) and fig (5) show the performance of the SCSI and IDE drives respectively. The plots show that in the SCSI case, read performance is almost unaffected with the increase in the buffer size. On the other hand, the size of the buffer matters for the IDE case. In the raw reads, data rate almost doubles with doubling the buffer size until it saturates around 20MB/s at buffer size of 16KB. The Ext2 performance is steady at 20MB/s for all buffer sizes.

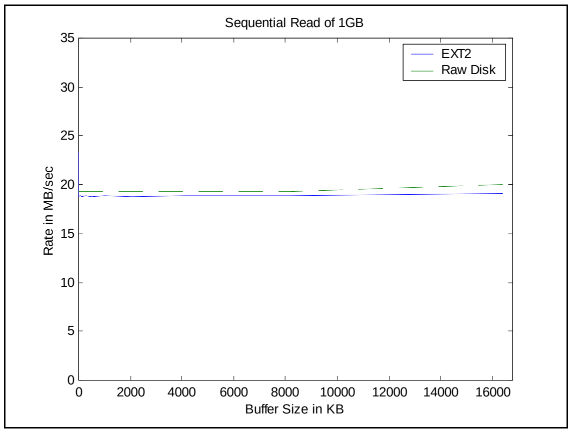

Figure 4. Sequential reads on SCSI with varying buffer size

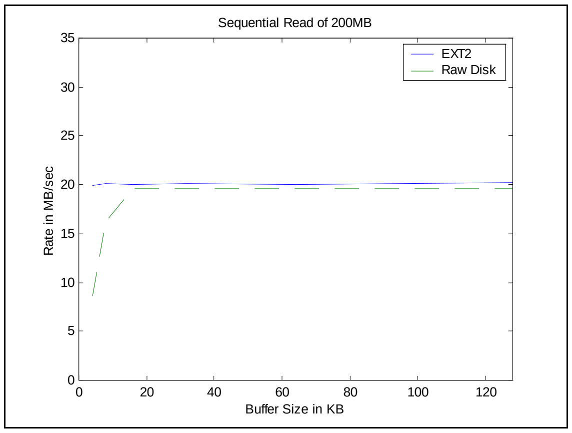

Figure 5. Sequential reads on IDE with varying buffer size

3.2.4. Simultaneous sequential reads

This benchmark measures the impact of simultaneous reads to two files in the ext2 file system. The benchmark starts by forking two processes. Each process opens a different file and reads a portion of it sequentially. The size read is again varied as described in section 3.2.1. The size of the buffer size being read is fixed at 1 MB. Fig(6) and fig(7) show the performance of the SCSI and IDE drives respectively. What should be expected is that the data rate will be almost equal and around half the full throughput of the data sequential rate of reads of the same size. In the SCSI case, this is true to a certain extent. Data rates for reading both files are catching up with each other throughout the experiment. The data rates increases steadily in the beginning as the buffer is warmed with pages of the two files, then, when the buffer starts filling performance drops. This happens at about 500MB of read size, which is about 60% of the estimated size of buffer cache for a single read (see section 3.2.1). The rate saturates towards the end at about 25% of the raw performance for a single read (5MB/s). The intuition suggests that the rate should be half the raw rate, the lost throughput can be attributed to the time spent seeking between the two files.

On the IDE machine, the same trend can be observed (bearing in mind the difference in memory size between the SCSI machine and the IDE one). The difference is that when both throughputs saturate, they do so at 50% of the raw rate. Again this can be attributed to the time spent in seeking between the two files. We included numbers for a similar experiment on the raw device, in which two processes simultaneously read from two distinct places on the device to imitate the behavior of reading two files using the file system. The results in fig(7) show that the two processes saw a steady data rate of 5MB/s for all read sizes which is 25% of the raw throughput and significantly less than the file system rate. We think the reason for this is due to the lack of knowledge on how Ext2 lays out files on the disk, so an honest imitation of the experiment on the raw device is hard to achieve and may result in varying seek times between the two.

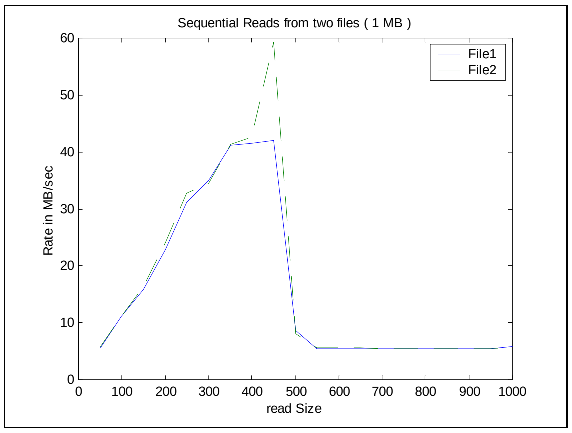

Figure 6. Simultaneous sequential reads on SCSI

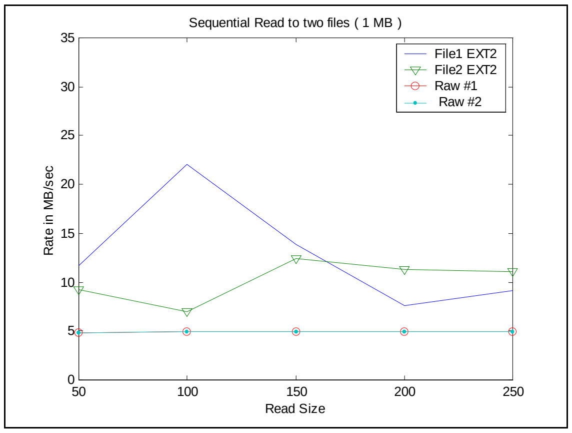

Figure 7. Simultaneous sequential reads on IDE

3.3. Write Benchmarks

3.3.1. Sequential write

This benchmark compares the performance of the Ext2 file system to the performance of the write on the raw device. The experiment settings are exactly as those in section 3.2.1. replacing the reads with writes and since there is actually no warming up for the buffer cache in writes, every experiment is run once. For the file system performance, we tested the case for overwriting files, recreating files and writing with the buffer cache disabled. In the SCSI system, as shown in fig(8), due to the effect of write behind cache in writes, Ext2 performance is superior to that of the raw device until around 400MB when it starts to fall down and catch up with the raw device rate. Disabling the buffer cache, however, will result in both Ext2 and file system being almost identical as shown in fig(9). It is also worth noting, that the performance in the case of overwriting files is slightly higher than in the case or recreation. We think this is due to the time spent deciding for file allocation.

Also worthy of notice that the raw write performance, being about 6.7MB/s, is significantly lower than the throughput in the raw reading case, which is about 20MB/s. Which makes the write rate about 33% of the read one.

Repeating the same benchmark on the IDE machine, resulted in a different behavior. In the case of sequential writes (with buffer cache disabled), the performance trend is almost the same as on the SCSI, with two main differences. First, the raw write performance is about 18MB/s which is comparable to the rate in the raw read case. Second, the gap between Ext2 writes and raw writes is much wider this time at about 82% of the raw throughput. Time couldn't permit more investigation as to the cause of this difference. Results of this experiment can be seen in fig(11). On the other hand, when the buffer cache was enabled (fig(10)), the performance results where drastically different. The rates in both recreating and overwriting the files in the Ext2 case were inferior to the corresponding raw writes despite the write-behind cache. We didn't have enough time to investigate the cause of this anomaly more.

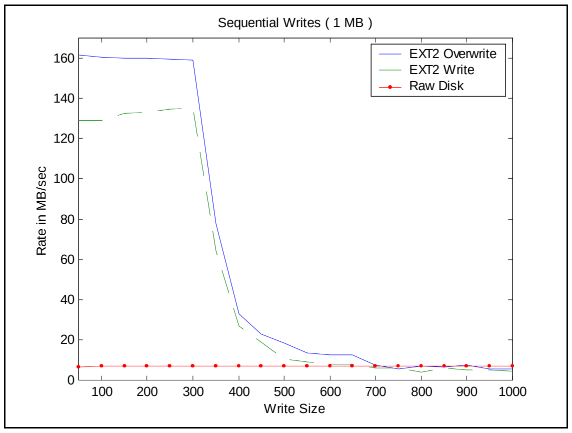

Figure 8. Sequential writes on SCSI

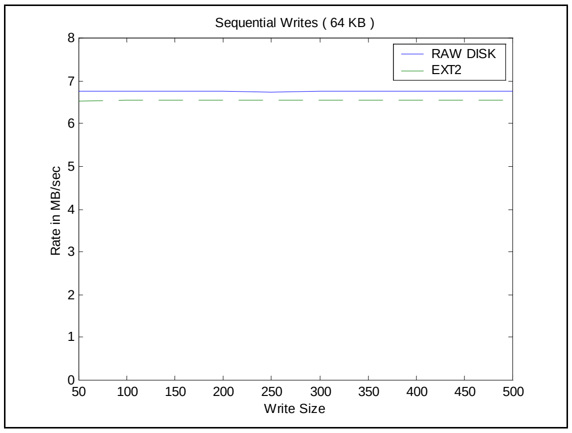

Figure 9. Sequential writes on SCSI with no buffer cache

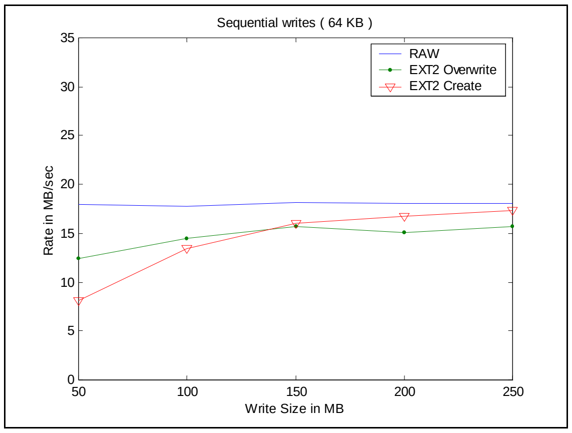

Figure 10. Sequential writes on IDE

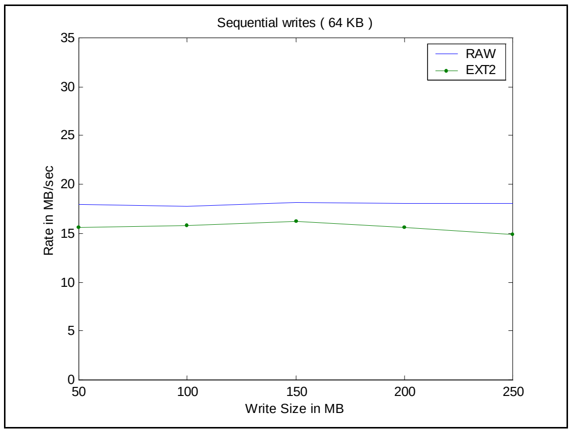

Figure 11. Sequential writes on IDE with no buffer cache

3.3.2. Buffer size effect on write performance

We studied the effect on varying the chunk size on the write performance. The settings for this benchmark are similar to those in section 3.2.3. for both environments except that we write a 200MB file for the SCSI machine. Fig(11) and fig(12) show the performance of the SCSI and IDE drives respectively. The plots show that in the SCSI case, write performance is almost unaffected with the increase in the buffer size. On the other hand, the size of the buffer matters for the IDE case. Data rate almost doubles with doubling the buffer size until it saturates around 20MB/s, which is the raw device performance. That is true for the ext2 file system as well as raw disk.

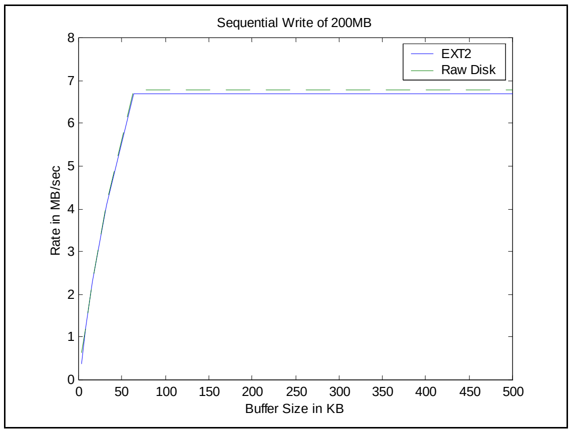

Figure 11. Sequential writes on SCSI with varying buffer size

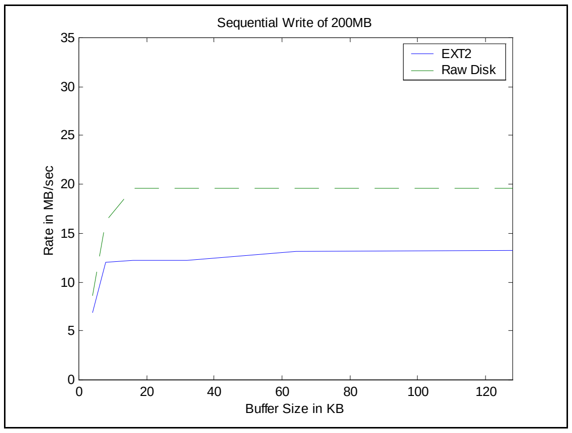

Figure 12. Sequential writes on IDE with varying buffer size

3.3.3. Simultaneous sequential writes

This benchmark measures the impact of simultaneous writes to two files in the Ext2 file system. The setting for the experiment is identical to that of the one in section 3.2.4. The only exception is that in the Ext2 case, the buffer cache effect is disabled. two plots have the same trend according to their available physical memory. Fig(13) and fig(14) report the results of the benchmark on the SCSI and the IDE machine respectively. The two plots have the same trends. In both, the write performance of the two processes is almost the same and slightly less than the corresponding raw performance of the same experiment. What is significantly different in the two experiments is the data rate observed compared to the raw write data rate in both settings. In the SCSI case, the data rate is about 60% the corresponding raw write rate. In the IDE case, however, the rate is about 2MB/s which is only 18% of the corresponding observed raw write performance (see section 3.3.1). We have no hint on what could be the cause of this performance drop, it might be worthy of further investigation.

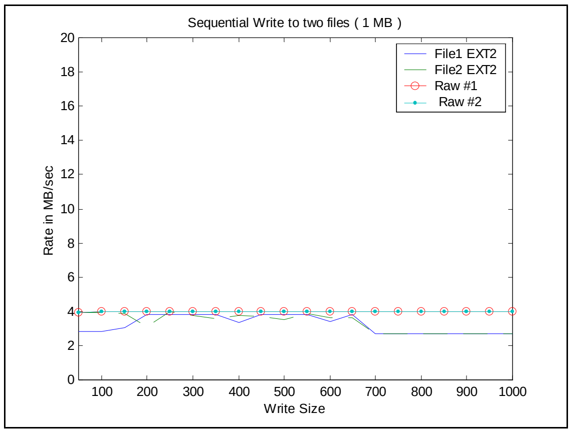

Figure 13. Sim. sequential writes on SCSI

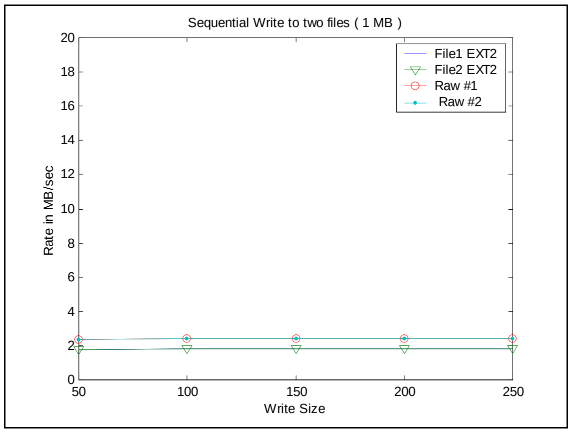

Figure 14. Sim. sequential writes on IDE

4. Conclusion

In this paper, we investigated a claim about a performance problem in the Linux Ext2 file system under streaming I/O. Our investigation was through a series of benchmarks that compared the read and write performance of the Ext2 file system against the raw performance of the underlying storage device. Our benchmarks were performed on two different environments, a machine from the DB cluster of workstations in the CS department of UW-Madison with 1GB of main memory and a SCSI interface and a PC with 128MB of main memory and an IDE interface.

The results of our benchmarks did not show any significant difference in performance between Ext2 and the raw device performance. In its worst case (i.e. without the help of the buffer cache), Ext2 performed at 82% of the raw performance in its worst cases (sequential writes on the IDE machine). Our benchmarks also show that there exists a wide gap in raw throughput on the SCSI machine between reads and writes which is not present on the IDE machine. The size of the buffer used to perform the reads or writes appears to have no significant difference on the performance of the sequential read in the SCSI case and the raw read performance on the IDE machine. In all other cases, the performance increases with the increase of the buffer size until saturation at the rate of the raw performance.

A side benefit of this paper is to serve as a report on the expected performance from the two settings on which our benchmarks were performed.

Further investigation into some non-intuitive results is required, especially in the case of the IDE machine.

5. References

[1] Remy Card, Eric Dumas, Franck Mevel. The Linux Kernel Book. Wiley, John & Sons, April 1998.

[2] Rémy Card, Theodore Ts'o, and Stephen Tweedie. Design and Implementation of the Second Extended Filesystem. *Proceedings of the first Dutch international symposium on Linux*.

[3] Peter M. Chen, David A. Patterson. A New Approach to I/O Performance Evaluation – Self Scaling I/O Benchmarks, Predicted I/O Performance. *Conference on Measurement and Modeling of Computer Systems* , ACM SIGMETRICS, 1993

6. Linux References on The Web

[4] www.linux.com

[5] www.linolium.com

[6] www.kernel.org

[7] www.lxr.linux.no

[8] www.ibiblio.org