On Benchmarking Popular File Systems

Matti Vanninen James Z. Wang
Department of Computer Science
Clemson University, Clemson, SC 29630
Emails: {mvannin, jzwang}@cs.clemson.edu

Abstract

In recent years, Windows, Linux and BSD have become dominant operating systems in personal computers [1]. An essential component of the operating system is its file system. Although there are reports [2, 3] on evaluation of file systems under the same operating system, it is necessary to assess the performance of different file systems under their associated operating systems to help selecting the operating system for different application requirements. In this paper, we evaluate six most commonly used file systems, NTFS, FAT32, Ext2, Ext3, ReiserFS, and UFS, in their respective operating systems. We compare their I/O performance under different workloads. Our results show that file system performance is largely tied to the cache and memory buffer management of the underlying operating system. Windows seems to support better buffering for random access of large files than Linux, but Linux based file systems offer faster sequential read/write performance. Journaling file systems trade some write speed for improved reliability. UFS performs consistently well for all I/O operations under any file sizes.

Keywords: Operating system, File system, Benchmark, I/O performance.

1 Introduction

People have blended computers into their daily life due to the exponential growth of the Internet and WWW applications. The popularity of personal computers has made Windows, Linux and BSD become dominant operating systems. Although some people select operating systems based on their applications, other people want to select an operating system with better system performance. A file system is part of modern operating system that provides permanent data storage with relatively low access latencies. The performance of many applications, such as database, is closely tied with the file system performance.

There are many factors that affect the file system performance. Those factors include disk block organization, file name mapping, metadata structure, reliability, concurrency control, and data searching algorithms. Besides the above data storage related factors, the cache and memory buffer management scheme in the operating system plays a very important role in system I/O performance.

Comparing the performance of different systems is a major challenge in computer science. Comparing products based solely on technical merits and specifications is rarely useful for predicting actual performance, so the most common approach is to use benchmarking software. In this paper, we use a benchmark tool developed by Iozone [4] to evaluate the I/O performance of six file systems, NTFS, FAT32, Ext2, Ext3, ReiserFS, and UFS, under three different operating systems, namely Windows, Linux and FreeBSD operating systems.

The rest of the paper is organized as follows. In section 2, we explain the criteria used for us to choose the benchmark. Then we briefly discuss the selected benchmark tool Iozone. In section 3, we discuss the test sets selection, test environment and the parameters used in the tests. We also briefly discuss the tested file systems in this section. We present the benchmarking results in section 4. Finally we will have our conclusion and discuss future works in section 5.

2 File System Benchmarking

There are many different approaches in evaluating file systems. Those file system benchmarking methods can be categorized into the following approaches:

- ***Workload based benchmarks:*** This approach emphasizes realism in benchmarking and relies on predefined workloads from real applications. Winstone [5], TPC [6], and SPEC [7] benchmarks belong to this category. Workload based benchmarks probably offer the best indication of "real world" performance, but often suffer from portability problems and make fair platform comparison infeasible. Furthermore, a workload based benchmark provides little to no information about which specific features of the system help or damage performance.
- ***Synthetic or kernel based benchmarks:*** This approach is designed purely to stress a particular aspect of system while attempting to minimize interactions with other parts of the system. Kernel based benchmarks use small parts of real applications, while synthetic benchmarks are programs specifically written for testing purposes. Iozone [4], Winbench [8], and Iometer [9] are examples of synthetic benchmarks. Synthetic benchmarks are typically more focused, and can be used to isolate performance bottlenecks and establish theoretical limits to performance.
- ***Trace-based analysis:*** This approach actually sets up tracing program in a real system, e.g., Windows NT 4.0 [3]. It analyzes the trace logs to evaluate the behavior and performance of the system. The study cycle for this approach is usually longer than the other approaches. It is very useful to find performance issues within one particular system and provides valuable information for improving the system. However it is not good for comparing the performance of different systems.

The objective of this paper is to perform a fair and meaningful comparison of different file systems on different operating systems using benchmark. An ideal file system benchmark should [10]:

- Give insight as to why a system performs as it does.
- Be I/O limited and scale well.
- Allow comparison across platforms.
- Be relevant to a wide range of applications.
- Be tightly specified and reproducible.

Based on these criteria, we select the benchmark tool Iozone [4] for our performance study.

Iozone is a purely synthetic benchmark which measures a variety of file operations. It has been ported to many operating systems and thus meets our requirement of evaluating different file systems under different operating systems. The program is easy to compile and run on both Windows and UNIX systems. Using Iozone, a wide range of tests are easily automated using command line parameters. Testing various file and record sizes is also straightforward. We will use Iozone to perform write, repeated write, read, repeated read, random read, random write, and strided read operations.

3 Tests Performed

We evaluate the performance of six file systems: FAT32, NTFS, Ext2, Ext3, ReiserFS, and UFS with soft updates. We test FAT32 and NTFS under Windows 2000, using the Cygwin library to provide a UNIX environment. We test Ext2, Ext3, and ReiserFS under Slackware Linux 8.0 (kernel 2.4.18), and evaluate UFS under FreeBSD 4.7. To make the paper self-contained, we briefly describe the tested file systems as follows:

- ***FAT:*** The File Allocation Table file system was invented in 1977 as a way to store data on floppy disks for Microsoft stand-alone Disk Basic [11]. Although originally intended for floppy disks, FAT has since been modified to be a fast and flexible system for managing data on both removable and fixed media. FAT currently can support a single disk volume up to 2 Gigabytes in size, with the increasing size of new hard drives this is an increasing problem. FAT32 is an enhancement of the File Allocation Table file system that supports large drives with improved disk space efficiency.
- ***NTFS:*** The New Technology File System [3] was originally developed for Windows NT and now is used in Windows NT, 2000 and XP. NTFS provides performance, reliability, and

functionality not found in FAT. NTFS includes security and access controls, encryption support, and has reliability control built in, in the form of a journaling file system.

- ***Ext2:*** The second extended file system is an extensible file system has become the most popular file system for Linux, and the default file system in most Linux distributions. Ext2 borrows ideas from previous Unix file systems using inodes to represent files and objects. It was designed to be extensible to make it possible to add features like journaling on at a later time.
- ***Ext3:*** The third extended filesystem is a journaling file system developed by Stephen Tweedie [12] as an extension to Ext2. It is mount compatible to Ext2 file system, but includes a journaling file to provide recovery capability. Ext3 can use all of the existing applications that have already been developed to manipulate the Ext2 file system. Journaling increases the file system reliability, and reduces recovery time by eliminating the need for some consistency checks.
- ***UFS:*** UFS is an UNIX file system based on BSD Fast File System (FFS) [13]. The term UFS refers to the implementation of FFS within the virtual node/virtual file system (vnode/vfs) framework which was first introduced by Sun Microsystems [14]. We use the FreeBSD version of UFS, which includes soft updates for recovery purposes.
- ***ReiserFS:*** ReiserFS [15] is the most mature and popular journaling file system available for Linux other than Ext3. ReiserFS is designed for performance and efficiency, particularly with small files, and incorporates features such as balanced trees for data storage, and non-block-aligned files

4 Test Results

We use Iozone version 3.153, compiled with GCC 3.2, for the benchmarking. The tests are performed on a clean 20GB file system containing nothing but the operating system. The computer used for testing is a PC with 800MHz Pentium III CPU, 16KB L1 and 256KB L2 cache, ATA-100 IDE controller, and 512MB RAM. The hard drive is a 7200RPM 30GB IDE ATA-100 drive with 2MB onboard cache and 8.9 ms reported average seek time. The hard drive is partitioned into a 20GB partition to house the tested file system and a 10GB partition for auxiliary data.

Sequential, random, and repeated reads and writes, and strided reads were tested. Record sizes of 8 KB-128 MB and file sizes of 64 KB-2 GB were tested. We report the test results in this section.

4.1 Sequential Operations

Transfer rates for sequential reads and writes are shown in Figure 1 and Figure 2 respectively. Reads are predictably much faster than writes, since the target pages might be already pre-fetched by operating systems. The three Linux file systems perform substantially better than Windows or FreeBSD when file sizes are smaller than CPU cache size. Otherwise, there is very little difference among read speeds for different file systems.

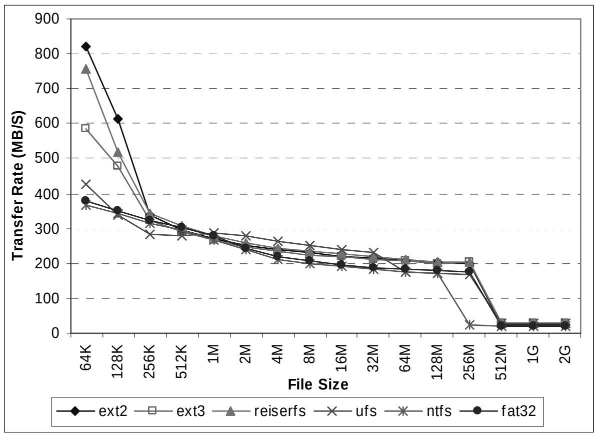

Figure 1: Transfer rates on various file sizes using sequential read operation.

Write speeds are more distinguished. Ext2 provides the best sequential write performance for mid-size files. Ext3 is a better journaling system than ReiserFS in terms of sequential write performance. Some operating systems provide write caching mechanism. Because write cache policy is critical for reliability and recoverability in some applications, a system which provides fast write access due to caching may not be preferable to one which does writes synchronously. Nevertheless, the superior performance of Ext2 over the more complex Ext3 and ReiserFS, and FAT32 over NTFS demonstrates the drawback to maintaining a journal and other additional metadata.

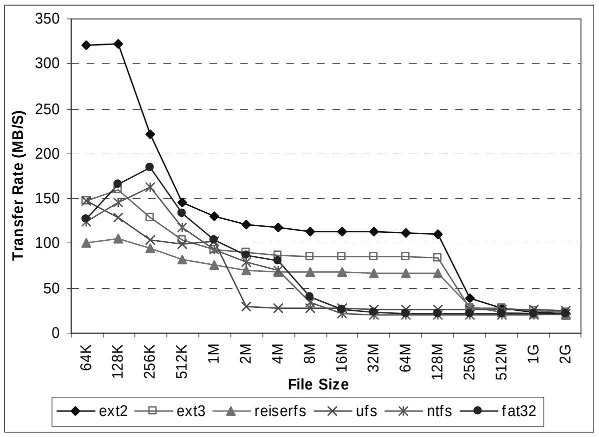

Figure 2: Transfer rates on various file sizes using sequential write operations.

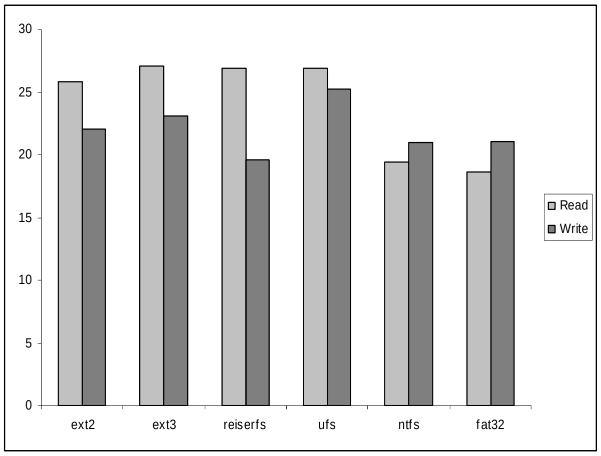

Figure 3: Sequential I/O transfer rates for various file systems with 2GB file.

We summarize the transfer rates for sequential file operations on 2 GB files in Figure 3. When file size is larger than main memory size, actual disk access is impossible to avoid. Thus this can be considered the most accurate indication of the comparative I/O performance of the file systems. When the file size is too small, the file system performance might be masked by the buffer management of the operating system. The weakness of FAT32 and NTFS in accessing disk is apparent here; neither performs competitively with the UNIX based file systems. On the other hand, Ext3 and ReiserFS show substantial differences in write and read speeds, due to the cost of maintaining the journal during updates. Despite adding the overhead of a journal, Ext3 performs better than Ext2 for reads and writes, demonstrating the design superiority of this file system. Overall, UFS seems to be the best performer, with high write and read speeds. This suggests that soft updates approach provides better performance in metadata management.

4.2 Random Operations

Transfer rates for random reads and writes are shown in Figure 4 and Figure 5. UFS offers the best random read and write performance. Both Windows file systems, FAT and NTFS, achieve similar transfer rates. The random I/O performance of the file systems under Linux is disappointing. Read speeds in file systems for Linux are about twice as slow as those in the other operating systems, and write speeds are even worse. This observation indicates it is necessary to improve the buffer management in Linux. In all file systems, random access performance was much better with small record sizes, which is not surprising due to the greater chance of an entire record being found in cache.

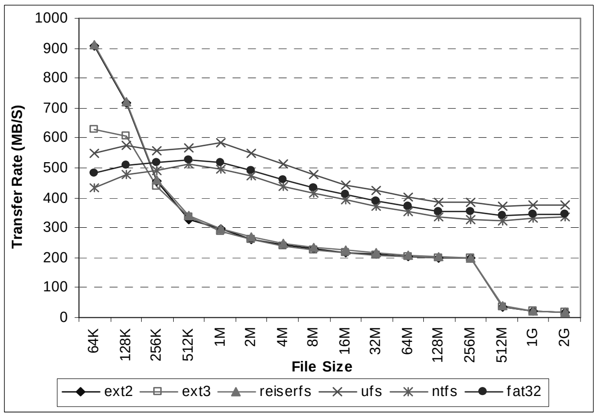

Figure 4: Transfer rates on various file sizes using random read operation.

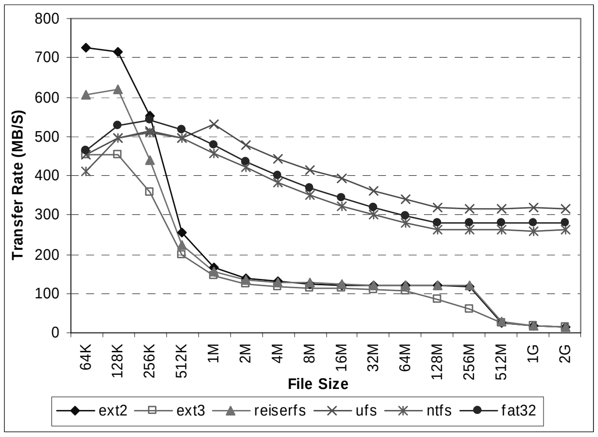

Figure 5: Transfer rates on various file sizes using random write operation.

We summarize the transfer rates for random file operations on 2 GB files in Figure 6. Once again read speeds are consistently higher than write speeds for all file systems. Clearly file systems for Linux are not efficient in handling large files. Their transfer rates are dwarfed by UFS, NTFS, and FAT32. On the other hand, Windows and FreeBSD are able to perform random I/O operations on files as large as 2GB without considerable loss in performance.

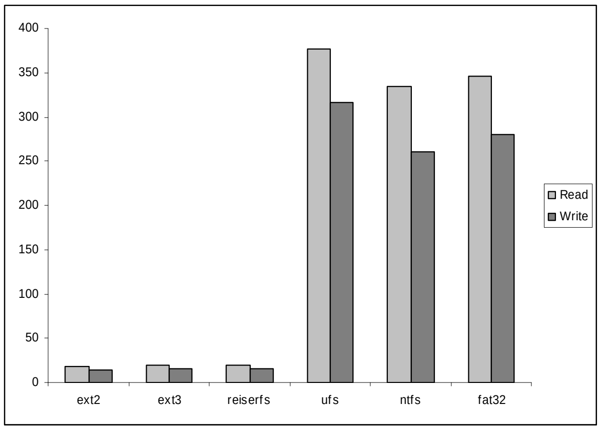

Figure 6: Random I/O transfer rates for various file systems with 2GB file size.

4.3 Repeated Operations

Figure 7 and Figure 8 show the performance of repeated reads and writes to a file. Repeated reads and writes are intended to test the effect of buffer management to the file system performance. As expected, repeated reads and writes are consistently faster than initial file access operations. However the difference is relatively small for read operations, because the operating systems normally pre-fetch data in sequential reads. The advantage of temporal locality is more apparent for write operations because operating systems can not prewriting data in initial sequential write operations. Repeatedly reading a small file in Linux is substantially faster than initial read. But the difference is not outstanding when the file sizes are larger. This observation and the other evidences lead us to believe the data searching mechanisms used in Linux file systems are excellent. However improvement has to be made in buffer management scheme due to its inefficiency of handling large files. The block allocation in Linux file systems is another issue that hurts its I/O performance in large files.

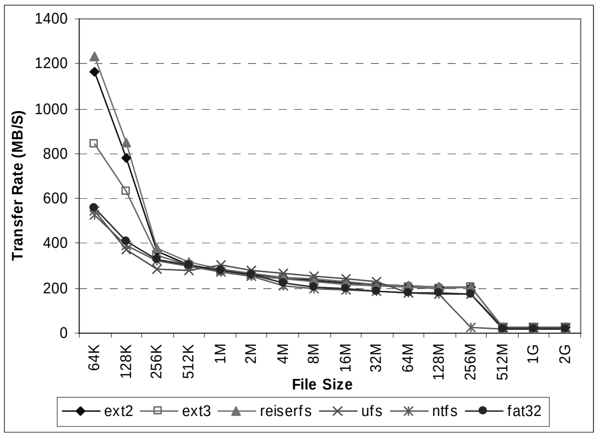

Figure 7: Transfer rates on various file sizes for repeated reads.

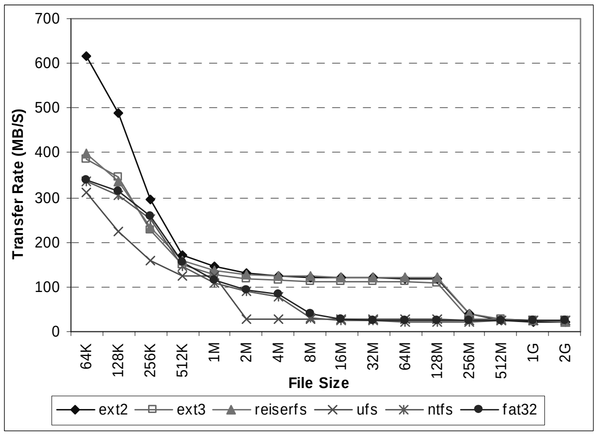

Figure 8: Transfer rates on various file sizes for repeated writes.

4.4 Strided Reads

We use Strided reads to test the effect of the record size to file I/O performance. The reason to use strided reads is to avoid the effect of pre-fetching by operating systems while maintaining the sequential disk operations. We report the transfer rates under various record sizes for strided reads on a 2 GB file in Figure 9. The figure shows transfer rates increase while the read record sizes increase. This is predictable because larger record sizes increases the amount of data per sequential read, and reduces the number of seeks within the file. However, when the read record size becomes too large, data transfer rates will not increase with the increase of record size. With very large record sizes the data transfer rates in FAT32 and NTFS declines while the transfer rates in the UNIX file systems stop increasing.

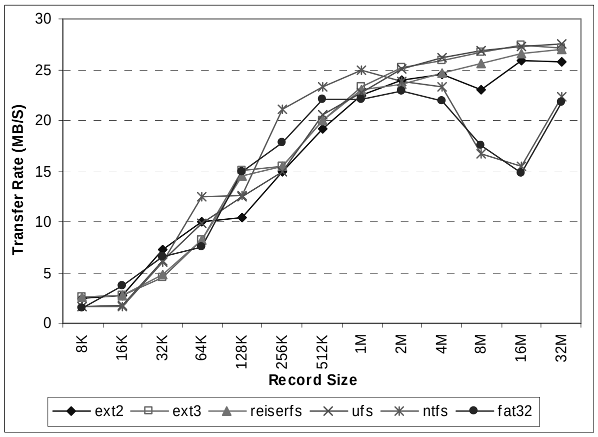

Figure 9: Transfer rates on various record sizes using strided read operation.

5 Conclusion and Future Works

In this paper, we evaluated six commonly used file systems, NTFS, FAT32, Ext2, Ext3, ReiserFS, and UFS, in their respective operating systems. We compared their I/O performance under different workloads. Our results show that file system performance is largely tied to the cache and memory buffer management of the underlying operating system. Buffer cache performance is the single most important factor in file system performance for most file sizes. Windows seems to support better buffering for random access of large files than Linux does. Linux based file systems offer faster sequential I/O performance. Journaling file systems trade some write speed for improved reliability. UFS performs consistently well for all I/O operations under any file sizes.

In most cases, differences in performance are much more dramatic across platforms than between file systems on a given platform. For instance, file systems under Linux operating system can not efficiently handle large files. As for the different file systems within the same operating systems, the simpler FAT32 and Ext2 file systems are generally faster but provide fewer features than their advanced alternatives. However, the performance advantage is minimal in most cases, and the additional features and robustness of NTFS or Ext3/ReiserFS make them the choices for their respective operating systems.

Our benchmarks were performed on a single computer with no other disk intensive processes. We plan to examine the performance of various file systems under concurrent workloads. In addition, we used the clean file systems as our test platforms. Usually an aged file system performs worse than a clean file system [16]. We will perform the similar tests on aged file systems.

References

[1] Carol Sliwa. Server Operating Systems, *ComputerWorld*, June 2001. http://www.computerworld.com/softwaretopics/os/

[2] Duc Vianney. Evalution of JFS, ReiserFS, Ext3fs and Ext2fs. *Linux Technology Center*. 2002.

[3] Werner Vogels. File System Usage in Windows NT 4.0. *in Proceedings of 17th ACM Symposium on Operating Systems Principles*. pp. 93-109, 1999.

[4] Don Capps and William Norcott. Iozone File System Benchmark, 1998. *http://www.iozone.org.*

[5] Business Winstone. Lionbridge Technologies, Inc. 2002. *http://www.etestinglabs.com/benchmarks/bwinston/bwinstone.asp*

[6] Transaction Processing Performance Council. *http://www.tpc.org*

[7] Standard Performance Evaluation Corporation. *http://www.spec.org*

[8] Business Winbench. Lionbridge Technologies, Inc. 2002.*http://www.etestinglabs.com/benchmarks/winbench/winbench.asp*

[9] Iometer disk benchmark. *http://sourceforge.net/projects/iometer/*

[10] Peter Chen and David Patterson. A New Approach to I/O Performance Evaluation – Self-Scaling I/O Benchmarks, Predicted I/O Performance. *In Proceedings of ACM SIGMETRICS*, pp. 1-12, 1993.

[11] Andrew Pitt. The FAT32 Resource page. *http://www.project9.com/FAT32/*

[12] Stephen Tweedie. Journaling the Linux Ext2fs Filesystem, *LinuxExpo '98*, 1998.

[13] Marshall K. McKusick, et. al. A Fast File System for UNIX. *Computer Systems*, Volume 2, Number 3, pp. 181-197, 1984.

[14] Filesystem Basics. *http://www.uwsg.iu.edu/edcert/session2/disk/filesystem.html*

[15] Hans Reiser. "ReiserFS v.3 Whitepaper." 2002. *http://www.namesys.com/*

[16] Keith Smith and Margo Seltzer. File System Aging—Increasing the Relevance of File System Benchmarks. *ACM SIGMETRICS*. 1997.