PenguinoMeter: A New File-I/O Benchmark for Linux®

Ray Bryant
Dave Raddatz
Roger Sunshine
Times N Systems
Austin, Texas
{raybry,daver,rsunshine}@timesn.com

Abstract

PenguinoMeter is a new open-source benchmark for Linux that measures file-system data transfer rates. PenguinoMeter allows the user to specify the file-system workload to be used in the benchmark in a very flexible manner. The workload specification is patterned after that of the Intel® Iometer benchmark; the current version of PenguinoMeter can read configuration files produced by Iometer. A series of comparisons between Iometer and PenguinoMeter is used to demonstrate that the workloads generated by these two programs appear to be identical. As an example use of PenguinoMeter, we compare the file system performance of Microsoft® Windows® 2000 Professional and Linux 2.4.9.

1 Introduction

To date, Linux has not been a significant player in the file system serving market. This is in large part due to the lack of availability of a good journaling file system. (*fsck*'ing a 50 GB file system after a power fault is not a strong selling point for most customers in this market segment.) However, this problem is being solved by a number of new journaling file systems that are either available as part of standard distributions or may soon be available (e.g. ReiserFS [ReiserFS], ext3 [ext3] IBM® JFS [JFS], and SGI™ XFS™ [XFS]). If Linux based file-servers are to compete in this market (as well as in the SAN and NAS markets), then good tools need to exist to enable comparative performance evaluation of the various Linux file systems as well as performance comparisons between Linux and non-Linux implementations. If we compare the benchmarks available for file systems on Linux (e. g. Bonnie [Bonnie], Bonnie++ [Bonnie++], IOzone [IOzone], PostMark [PostMark]) to those available for under Microsoft® Windows® (e. g. Iometer [Iometer] from Intel Corporation) it is clear that the latter is far more sophisticated than the former.

In this paper, we present the design and implementation details of a new file-I/O benchmark for Linux. We call this benchmark "PenguinoMeter"[1] (or "pgmeter"). We describe *pgmeter* as a file-I/O benchmark instead of a file-system benchmark because we are principally interested in measuring the rate at which data can be transferred to and from the file system as opposed to measuring the rate at which file-system operations (such as file creation or directory lookup) can be performed. Since our company is in the business of producing a storage solution for Windows and Linux, we are also motivated by the need to provide a good benchmark to evaluate and improve the performance of our company's hardware and software products under Linux.

In the Windows environment, a popular and sophisticated file-system benchmarking tool called Iometer is available from Intel Corporation. *Pgmeter* is designed to support the same workload-modeling framework that has made Iometer so successful in the Windows arena. *Pgmeter* provides support (with some restrictions) for reading workload configuration files produced by Iometer. Thus one can use Iometer configuration files developed by third parties (see the Iometer discussion in Section 2 for examples) to test the same kind of workloads under Linux that have previously been tested under Windows. Of course, this is only interesting if it can be demonstrated that Iometer and *pgmeter* create similar workloads when given the same configuration file as input.

To show that this is indeed the case we ported *pgmeter* to Windows 2000 using the Cygwin™ library [Cygwin]. We then ran Iometer and the (ported) version of *pgmeter* on a representative sample of workload configuration files. The maximum difference between the reported results was 2.5% in the 78 trials we conducted. (See Section 5 below for details.). Thus we believe it is reasonable to say that both programs

[1] The preferred pronunciation is "pen-GWYN-oh-meter"; the pronunciation "pen-gwyn-OHM-eter", while deprecated, is also acceptable.

simulate the same kind of workload when given the same input data.

Of course, this experiment does not demonstrate that *pgmeter* under Linux simulates the same workload that Iometer does under Windows 2000. This is especially true since Iometer (and our ported version of pgmeter) bypass the file-system cache under Windows [Iometer]. However, it is still an interesting comparison to run *pgmeter* on the two systems. Since Linux has no way to bypass the file-system buffer cache analogous to that which exists under Windows, we resort to a traditional technique of minimizing the impact of the file-system cache on the results: We use a file that is more than twice as large as the main memory on the system. Results of these experiments are presented in Section 5.

While we have tried to make *pgmeter* simulate a workload that is as similar as possible to that of Iometer, there is one significant difference between the two programs. *Pgmeter* is an open-source program that is freely distributed under the GPL. (Intel has chosen not to distribute the source for Iometer; also Intel [IntelPers] does not currently plan a port of Iometer to Linux.) A SourceForge™ [SourceForge] project for *pgmeter* has been established at pgmeter.source-forge.net. It is our intention to make source code for *pgmeter* available for downloading from that site by the time this paper is published. We welcome the participation of the open source community in the *pgmeter* project. Further details of this participation will be organized through pgmeter.sourceforge.net.

In the next sections of this paper, we first examine the existing file-system benchmark programs and explain why we feel a new benchmark is required. Next, we describe the workload-modeling framework of Iometer and summarize other significant implementation details available from the Iometer FAQ. We then discuss how we implemented a corresponding workload in *pgmeter*. Following this, we describe the benchmark comparisons we have performed between Iometer and the Cygwin version of *pgmeter*. We also compare *pgmeter* running under Windows 2000 to *pgmeter* running under Linux 2.4.9. This analysis demonstrates that Linux file system performance is in many cases quite good. Finally, we discuss the current status of the program and directions of future work.

2 File-System Benchmarks

In this section, we discuss a number of file system benchmarks available under Windows and Linux. This list is largely taken from the list of I/O benchmarks available at:

http://www.acnc.com/benchmarks.html

However, we have only included those benchmarks whose Web pages appear to be current, and we have deliberately eliminated benchmarks designed for DOS or that test specific hardware characteristics of the disk subsystem. We have also eliminated benchmarks that appear to be obsolete, unused, or undocumented, or that do not support a variety of disk workloads.

For our interests, we prefer a benchmark that has the following characteristics:

- It should be open source and readily portable to Linux.
- It should support a flexible and realistic workload specification that allows a user to model a number of different workloads.
- It should be file-I/O (read and write) as opposed to file-system operation (file creation and deletion, directory operations, etc.) intensive. The latter type of test is more influenced by the design of the file system itself as opposed to the underlying block-device software and hardware that we are interested in testing.

The second point above implies that we are not interested in trace-driven benchmarks. Trace-driven benchmarks can be very representative of a particular workload, but they are cumbersome and difficult to modify to represent workloads other than that of the traced system. For example, it is difficult to modify a trace-driven benchmark to answer such questions as "What would happen if the average request size was increased by 20%?". We thus prefer synthetic workloads that depend on a small number of parameters to characterize the workload to the trace-driven workload approach.

With the above in mind, here are the other benchmarks we considered:

Bonnie and Bonnie++

These two programs are probably among the most well known and widely used file system benchmarks. They are simple and easy to use and can be readily ported to new operating system environments.

Bonnie [Bonnie] measures the rate of writing and then reading a file using

- *Putc()* and *getc()*
- "Efficient" block I/O operations
- Rewriting the existing file

Bonnie then creates 4 child processes that seek randomly in the file that has been created, read the block found there, and in 10% of all such reads, rewrites that block. The rate of executing this sequence is reported in operations per second. In all cases, Bonnie also reports the CPU busy time during each such test.

Bonnie++ [Bonnie++], aside from being written in C++, adds the following capabilities:

- Bonnie++ supports access to data sets larger than 2GB using multiple files.
- Bonnie++ includes tests of the rate at which the file-system can create large numbers of files.

The principal drawbacks of both of these programs are that they are not fully parameterized in terms of workload characteristics and they do not allow mixture workloads (mixtures of random and sequential accesses, for example) to be represented.

NetBench ®

NetBench [NetBench] is a product of Ziff Davis Media, Inc. It is designed to emulate the load a network of Windows PC's would place on an SMB file server. The workload script can be edited and customized, but the number of clients required to fully load a server can be large. Also, since this requires Windows clients and an SMB file service, it does not really address our need of benchmarking native file-systems under Linux.

WinBench ®

WinBench is also a product of Ziff Davis Media, Inc. Although it tests other subsystems as well, it includes a disk subsystem performance test. This test is designed to emulate the disk behavior of a mixture of Windows applications running on an office PC. This is a trace-driven workload with several components obtained from traces of a system running office applications and the Windows operating system itself; a sequential transfer test is also included. Since this is both a closed-source and a trace-driven benchmark, it does not meet our needs.

Dbench

Dbench [Dbench] is a GPL benchmark intended to emulate the workload created by the Netbench benchmark. Dbench replays a set of around 90,000 requests that were captured by a network sniffer during an execution of Netbench. It does not require a large network of clients, but it is typically run on a system that is network-attached to the server system. While Dbench eliminates the need for Windows client machines, it still issues SMB requests so it requires a Windows server or a Samba [Samba] server running under Linux. Since we are primarily interested in benchmarking native file systems under Linux, this benchmark does not meet our requirements. Also, this is a trace-driven benchmark so it does not meet our flexibility requirements.

IOzone

IOzone [IOzone] is a highly portable file-system benchmarking tool. It supports a variety of file mode accesses including sequential, random, read, and re-read as well as a number of file system interfaces such as normal and asynchronous I/O. It also supports multiple threads of execution (on the local system). However, IOzone does not support variable request sizes, nor does it support mixtures of reads and writes or mixtures of sequential and randomly sequenced requests[2]. In fact, IOzone is probably better characterized as a file-system tuning tool instead of being a benchmark that can model a file system workload. IOzone is designed to measure file-system performance at a variety of I/O request sizes and to produce a report showing the relationship between throughput and I/O request size. Because of its inability to represent mixture workloads, IOzone is unable to model complex workloads such as an OLTP file-system workload. However, because it is well known, source code is available, and it has already been ported to Linux, we have actually used this benchmark in some of our performance studies.

PostMark

The PostMark [PostMark] benchmark was developed to model the "small-file" workload that is encountered when supporting electronic mail and news services with many users simultaneously accessing the data. As such, it is file-system operation intensive (create small file, delete small file, directory lookup, insert, delete, etc) instead of being file system transfer intensive. This means that PostMark tests file system operations instead of testing the I/O capabilities of the file system and its underlying hardware. We are more interested in the latter than the former so this benchmark does not meet our needs.

Nbench

NBENCH [Nbench] is a Windows NT and Windows 95 benchmark. It measures disk performance by writing and reading a 10 MB file sequentially. It uses the Windows file attribute FILE_NO_BUFFERING to keep the operating system from caching the file. Multiple threads are supported, but must be targeted to different physical disks to avoid seek interference

[2] IOzone "telemetry" files can be used to model mixtures of random and sequential accesses but this is an example of trace-driven workload specification.

between the files. Since this benchmark does not support a customizable workload, and since source code is not available, it is not appropriate for our work on Linux.

NTIOGEN and IOGEN

NTIOGEN [NTIOGEN] is an NT port of the Symbios Logic Unix benchmark IOGEN. It creates a number of child processes that then generate read and write requests against either a physical drive, a partitioned device, or a given file. While one can change the number of child processes (and hence the number of outstanding I/O's) it is not apparent how to modify the workload in any other way.

Iometer

Intel's Iometer [Iometer] includes a flexible workload-modeling framework. This framework allows the user to specify a mixture of read and write accesses, a mixture of sequential and random accesses, as well as a mixture of request sizes. Additionally, Iometer supports local and remote worker threads. Like Nbench, Iometer uses the Windows file attribute FILE_NO_BUFFERING to avoid file system cache effects while running the test (see the FAQ available with Iometer). Iometer thus primarily measures the performance of the disk subsystem and is often used to evaluate the performance of different disk devices or SAN or NAS storage devices (see, e. g. [StoReview], [Etesting], [Dvault], [NetC], [AdRaid], or [NWF]).

The widespread use of Iometer has also lead to the standardization of certain workloads. For example, [StoReview] uses the following Iometer workloads:

- File Server: This workload is a mixture of 512 byte to 64 KB transfers with the most popular transfer size being 4 KB and the mean transfer size being 11300 bytes. (The complete definition of this workload is given in the Appendix.) Of these transfers, 80% are reads and 100% are random accesses.
- Workstation: This workload consists of 8 KB transfers of which 80% are reads and 80% are random accesses.
- Database (which we call OLTP in this paper): This workload is defined as 8 KB transfers, of which 67% are reads and 100% are random accesses.

Intel distributes all of these workloads with Iometer except for the Workstation workload; Storage Review created the latter. Another workload distributed with Iometer is a Web Server workload that is a mixture of 512 byte to 512 KB transfers; with 4K being the most popular size, and the average transfer size being 16035 bytes. (A complete definition of this workload is also given in the Appendix.)

Now whether these workloads are actually representative of their namesakes is not really the point. Instead the important point is that such workloads exist and are readily available for comparison purposes. Thus, one can compare the "file system" workload on different systems and publish such a result and readers should be able to find out what that workload means.

Except for the fact that Iometer is not an open-source benchmark, it meets all of the rest of our requirements. Another reason that Iometer is interesting to us is that we have made extensive use of Iometer in testing our Windows products; it made perfect sense for us to desire to have a similar program to assist in evaluation of our products for Linux

A port of Iometer to Linux was at one point supposed to be underway at Intel, but our most recent information on this is that this effort has been abandoned at the present time [IntelPers]. For this reason, we decided to develop a program with similar capabilities for Linux and to make the source code for this new program readily available.

3 The Iometer Workload Specification

Iometer defines the file system workload in terms of "workers" and "access specifications". Each worker can have multiple access specifications assigned to it. For a particular benchmark run, only one of these specifications is active. This allows a series of benchmark runs to be defined by assigning multiple workload specifications to a particular worker.

Each access specification consists of a number of "access lines". An access line defines the following quantities:

Size: The I/O size for this access line.
Percent: The fraction of the I/O's generated by this access specification that are generated by the parameters of this line.
% Read: The fraction of the I/O's generated according to the parameters of this line that are read requests.
% Random: The fraction of the I/O's generated according to the parameters of this line that are random requests.
Delay: Delay after burst count of I/O's have completed.
Burst: The number of I/O's in a burst request.
Alignment: The alignment of the I/O requests.

There can be an arbitrary number of access lines per access specification. For example, see the "Web Server" and "File Server" workloads described in the Appendix.

We interpret the "% Read" and "% Random" parameters as follows: "% Read" is the fraction of requests generated by an access line that are read requests. "% Random" gives the probability that the next I/O request will be randomly selected from all the possibilities available in the output file. With probability 100%-"% Random" the next request generated according to this access line will be for the next sequential record in the file.

The workload specifications used by Iometer can be created and saved to an .icf (Iometer configuration file) for later use in another run of Iometer. This file is stored as ASCII text and is (with some practice) human readable.

Another component of the Iometer workload specification is the number of outstanding I/O's to keep active against the target file (or device). Iometer uses the asynchronous I/O capabilities of Windows 2000 to implement this feature.

The Iometer workload specification allows the file system target of the benchmark to be specified as a file or a Windows disk partition (e. g. "all of D:").

Iometer supports a number of different iteration modes that control how to run a series of benchmark experiments. For example, one can specify that the benchmark be repeated for 1, 4, 16, 64, and 256 outstanding I/O's. Also, if multiple access specifications are assigned to a particular worker, one can request that the benchmark be repeated for each such access specification. This iteration can be combined with the iteration on outstanding I/O's if desired.

Iometer also supports remote disk workers (workers on machines network attached to the machine where Iometer is running) as well as network targets that can be used to measure network bandwidth.

4 The Pgmeter Implementation

Pgmeter supports a set of features similar to a subset of the facilities of Iometer. In particular, *pgmeter* can read and parse an Iometer configuration file (.icf file) provided it meets the following restrictions:

1. Only file system targets are supported. No network targets or disk partition targets are supported.
2. Only local disk workers are supported. *Pgmeter* does not support remote disk workers.
3. The only methods of iteration that are supported by *pgmeter* are "Normal" or "Cycle_Outstanding_IOS". These two modes include the iteration mode described in the last section; these modes are the ones we have found most useful in our studies.

Unlike Iometer, *pgmeter* allows the user to specify the target file name on the command line. (The target, or data, file name is the file to and from which the data transfer rate is measured.) Thus, the restriction on disk targets is minor since the effect of a disk partition target can be achieved under *pgmeter* by specifying as the file name to the *pgmeter* command the block device name of an entire partition.

Also not implemented in *pgmeter* are a wide range of features supported by Iometer, including, a graphical configuration user interface, the ability to create, modify and save configuration files, and a fully graphical display of results. (For our purposes under Linux, we have found "vi" to be an acceptable substitute for the graphical configuration interface of Iometer, since in most cases we are simply modifying an existing .icf file.)

Since Linux does not directly support asynchronous I/O (multiple threads are used inside of glibc to support the POSIX AIO interface), and because we wanted to port our code to Windows using the Cygwin library (where AIO is not supported), we decided to simulate the outstanding I/O's facility of Iometer using multiple processes.[3] However, this had to be done carefully in order to simulate the same workload as Iometer. If, for example, one were to merely create N processes to simulate N outstanding I/O's and not carefully coordinate them, then random disk head movement can occur that will dramatically change the observed I/O rate and I/O bandwidth.

To solve this problem, a global "current sector address" (or CSA) is maintained for each file in a *pgmeter* test. Each child process updates the CSA for its file according to the following algorithm:

1. Choose a line from the access specification according to that line's probability.
2. Read the global CSA.
3. Using the "% Random" field from the chosen access line as the probability of choosing random

[3] Due to the similarity of processes and threads under both Linux and Cygwin, we will also use the term threads in the rest of the paper.

access, make a random choice to determine if this access is to be a "random access" or a "sequential access". For sequential access go to step 5. Else go to Step 4.

4. (Random access.) Choose a random seek point as a new-CSA and exchange and swap that new value into the CSA. If the exchange and swap fails, reread the CSA, choose another new-CSA seek point in the file and try again until success. Go to step 6.
5. (Sequential access.) Using the "Size" field of the current access line, calculate a new-CSA by adding the "Size" field (converted to sectors). If this new address is past the end of the file, set the new-CSA to sector 0 of the file. Exchange and swap the new-CSA into the CSA. If the exchange and swap fails, reread the CSA, calculate a new value for the new-CSA and try again until success.
6. Using the "% Read" field from the chosen access line as the probability of choosing read access, make a random choice to determine if this access is to be a read or write.
7. Seek to the sector address new-CSA of the file and do the read or write request for "size" bytes.

In this way, if an access line specifies 100% sequential accesses (0 percent random) each process will request the next record in the file in order. Of course, this is an approximation since after a process has selected a record in the file to read, that process may be suspended before it is able to issue its I/O request. Nonetheless, as we shall show in the next section, this technique is sufficiently good that it allows *pgmeter* to simulate the Iometer workload to a high degree of accuracy.

Internally, *pgmeter* is divided into four modules: (1) the configuration module, (2) the display module(s), (3) the worker module and (4) the monitoring module.

The configuration module is responsible for parsing the configuration files. At the present time, only configuration files generated by Iometer are supported. (Ultimately *pgmeter* will also support configuration file creation and editing and we envision defining a new *pgmeter* configuration-file format.) During parsing, the configuration module builds the configuration data structures used to drive the remainder of the program

The display modules allow encapsulation of different methods of displaying real-time execution results. At present, the only available display module supports a curses-based bar-graph representation of the data. A Gtk based strip-chart display module is under development, but is not presently available.

The worker module is where the real work of *pgmeter* is performed. This module reads the configuration data structures created by the parser module and runs the tests described there. The worker module includes the threads that actually perform the I/O. The worker threads also place individual statistics into the shared memory area for collection by the monitor module.

The final module is the monitor module, which is responsible for starting and stopping the worker threads, monitoring their execution, calling the display module(s) with updated information and printing results.

5 Benchmark Results

In this section we report the results of two benchmark experiments:

1. A series of experiments run under Windows 2000 Professional comparing Iometer and a version of *pgmeter* ported to Windows using the Cygwin[4] library.
2. A series of experiments that compare *pgmeter* running under Cygwin (and Windows) to *pgmeter* running under Linux 2.4.9.

The purpose of the first series of experiments is to demonstrate that *pgmeter* appears to synthesize the same disk I/O workload that Iometer does. The purpose of the second series of experiments is to provide an example of the utility of *pgmeter* in file system benchmark studies and to provide a comparison of the performance of the file systems of Windows 2000 and Linux.

Iometer and Pgmeter Comparison

As mentioned before in this paper, Iometer uses the FILE_NO_BUFFER attribute when creating or opening its data files. This attribute causes Windows to bypass use of the file-system cache when accessing the file. In order to provide a fair comparison we modified the Cygwin library to support a new flag on the Cygwin *open()* call: O_NOBUFFER. When this flag is encountered during a Cygwin *open(),* it causes the library to include FILE_NO_BUFER on the call to the Windows routine *CreateFile().* P*gmeter* was then modified to provide a command-line option to control opening of its test files using this new flag. In this

[4] Cygwin is a Unix environment for Windows. Here we are using the Cygwin library that provides a Unix emulation layer, which allowed us to easily port *pgmeter* to Windows.

way, both programs use the same operating system interface to access their data files.

Iometer was used to create the data files, and both programs use the same data files. This avoided any bias due to placement or fragmentation of the target files. The disk defragmentation tool of Windows 2000 Professional was used to make sure that the data file was a single extent and was not fragmented in any way. (The data file was 1GB in size). The data file was on its own physical volume with the Windows operating system on another physical volume in order to reduce operating system interference as much as possible. The data volume was formatted with NTFS. Finally, the Windows performance monitor tool was used to measure disk data transfer rates and disk request sizes to make sure that these matched what we expected the programs to generate.

Three trials were conducted on three identically configured machines (See the Appendix for machine descriptions). A total of 78 different test cases were run for each trial under Iometer and pgmeter respectively. (A test case is one benchmark run; running a particular access specification with 3 different outstanding I/O counts is considered 3 trials). A complete list of the test cases used is given in the Appendix.

The principal statistic examined for the sequential tests was data bandwidth in MB/s. The principal statistic for the random and mixed tests (web server, file server, OLTP) was number of I/O's per second.

At the end of the test, the statistics for the three trials from each test case were averaged and corresponding average results were compared. In all of the trials the maximum difference between the average statistics for Iometer and pgmeter was 2.5%. We thus feel confident in stating that pgmeter simulates the same workload that Iometer simulates.

One limitation of this testing technique is that pgmeter under Cygwin cannot support more than 63 outstanding I/O's. This is a limitation of the Cygwin library since it depends on using Windows wait objects to implement *wait*(). Thus it is impossible to have more than 63 child processes in Cygwin. For our tests we typically use a geometric sequence of outstanding I/Os (1, 2, 4, 8, 16, etc) so for the pgmeter to Iometer comparison tests we capped the number of outstanding I/O's at 32.

Windows 2000 and Linux 2.4.9 Comparison

Having demonstrated the equivalence of pgmeter and Iometer under Windows, we next decided to compare file-system performance under Windows and Linux by running *pgmeter* on each system. Unfortunately, there is no analogue to the FILE_NO_BUFFER flag under Linux. Additionally, there is no good way to implement such a flag in Linux. This is because the implementation of the Linux file-system is closely tied to the behavior of the file-system cache. For example, while it possible to write a simple system call to flush a page from the file-system cache before reading the page, the result of this action is to defeat the read-ahead logic in the kernel. This results in very poor file-system read performance.

Given this, we decided to fall back on a time-honored technique for file-system performance measurement: On each system, we used a file that was several times larger than the main memory of the machine. In this case the machines had 512 MB of RAM and we used a 2 GB file. Also, the FILE_NO_BUFFER flag was not used in *pgmeter* during this sequence of Windows benchmarks. These rules were designed to provide a fair as possible comparison between the two systems.

Four identical machines were used for these tests; descriptions of the machines are given in the Appendix. The data file for the test was again on a dedicated physical disk and the operating system was on a separate disk. On the Windows systems the file was created on an otherwise empty NTFS partition (except for the ever-present Windows system files). On the Linux systems the data file was created on a newly *mkfs*'d ext2 file system partition. As before, 3 trials of each case were completed and we report here statistics that are the average of the statistics from the 3 trials. Each case consisted of a 30 second warm-up period followed by a 5 minute run for data collection.

To further ensure that file-system cache effects did not bias the results, the systems were rebooted after the sequential trials had completed and again after the random trials had completed (i.e. before the mixed tests such as OLTP, file server, workstation, and web server were begun). We also repeated the first two cases of each series of runs at the end of the series. We saw no significant difference between those repeated cases. This indicates that caching effects during a series between reboots were not significant.

Comparisons between the two systems are reported in Figures 1-6. In Figure 1 (sequential read tests), we see that Linux provides better read performance at 4 and 16 threads (outstanding I/O's), while for 1 thread Linux and W2K are pretty much equivalent except at the 64 K record size where W2K performance falls off significantly. In Figure 2 (sequential write tests) we see that W2K is around 10% better than Linux and that performance is insensitive to either the number of outstanding I/O's or the record size. Figure 3 (random read tests) shows that Linux and W2K are more or less equivalent at 1 and 4 threads, but that Linux is faster at 16 threads. In Figure 4 (random write tests) we see that W2K is faster than Linux except for the smaller record sizes and 1 thread. Linux on the other hand, seems to be relatively insensitive to the number of outstanding I/O's and the important factor in determining the I/O rate is the record size. Finally, in Figures 5 and 6 we see that Linux consistently outperforms W2K for the 8K OLTP, Workstation, File Server, and Web Server workloads.

Of course, ext2fs is not a journaling file system, while NTFS is, so in that sense these comparisons are not completely fair. For many applications, the slight performance disadvantage of a journaling file system is outweighed by its improved recovery times in case of system failure. Further comparisons of this type should be performed using a journaling file system under Linux.

6 Status and Future Work

At the moment, the interface to *pgmeter* is a command-line interface only. There is a rudimentary graphics output using curses. Multiple threads of execution are supported on the system where *pgmeter* is executing; remote worker threads for simultaneous benchmarking of multiple servers are not yet supported.

In the future, we expect to develop a GUI for *pgmeter* and fancier graphics output. Support is not currently planned for remote-worker threads or network tests. However, we are planning on making *pgmeter* available as open source and an open source project for this purpose has been created at pgmeter.sourceforge.net. Perhaps someone in the open source community will decide to add additional functionality to *pgmeter* such as remote workers, network tests, or a nice GUI.

The Linux 2.4.9 ext2 to Windows 2000 NTFS comparisons indicate that while Linux performs better for many of the read-intensive cases, there is room for improvement in the sequential and random write cases. Of course, this is a comparison between a non-journaling and a journaling file system. Further comparisons of this type should be done using a journaling file system under Linux.

7 Acknowledgements

The authors would like to acknowledge the support and assistance of the management at Times N Systems in the creation of this paper. We additionally acknowledge the contributions that VA Linux, Inc. has made through the SourgeForge project.

8 References

[AdRaid] http://www.adaptec.com/pdfs/3210S_vs_mylex352_final.pdf
[Iometer] http://developer.intel.com/design/servers/devtools/Iometer/index.htm
[Bonnie] http://www.textuality.com/bonnie
[Bonnie++] http://www.sourcepole.com/sources/reviews/raid
[Cygwin] http://www.cygwin.com
[Dbench] http://www.samba.org
[Dvault] http://www.dell.com/us/en/esg/topics/power_ps4q00-power.htm
[Etesting] http://www.etestinglabs.com/main/reports/baiomtr.pdf
[ext3] http://www.us.kernel.org/pub/linux/kernel/people/sct/ext3
[IntelPers] Personal Communication.
[Iometer] http://developer.intel.com/design/servers/devtools/iometer/index.htm
[IOzone] http://www.iozone.org
[JFS] http://oss.software.ibm.com/jfs
[Nbench] http://www.acnc.com/benchmarks/ntiogen.zip
[Netbench] http://www.etestinglabs.com/benchmarks/netbench/netbench.asp
[NetC] http://www.networkcomputing.com/1209/1209f28.html
[NWF] http://www.nwfusion.com/reviews/2000/0821revhow.html
[PostMark] http://www.netapp.com/tech_library/3022.html
[ReiserFS] http://www.reiserfs.org
[Samba] http://www.samba.org
[StoReview] http://www.storagereview.com
[XFS] http://oss.sgi.com/projects/xfs

Appendix

Workload Descriptions

For all of the workloads used in this paper, the delay parameter is zero and the burst size is one for all access lines of this access specification. The alignment specification is 0 for all of the access lines; this implies sector-level alignment.

OLTP

This workload is a single record size (8 KB) with 67% reads and 100% random access.

Workstation

This is the Storage Review.com's workload. It is defined as 8KB records, 80% reads, 80% random access.

File Server

The File Server workload uses the following request size distribution:

Size	Percent	% Read	% Random
512	10	80	100
1024	5	80	100
2048	5	80	100
4096	60	80	100
8192	2	80	100
16384	4	80	100
32768	4	80	100
65536	10	80	100

Web Server

The Web Server workload uses the following request size distribution:

Size	Percent	% Read	% Random
512	22	100	100
1024	15	100	100
2048	8	100	100
4096	23	100	100
8192	15	100	100
16384	2	100	100
32768	6	100	100
65536	7	100	100
131072	1	100	100
524288	1	100	100

Machines Used for the Benchmarks

For the *pgmeter* to Iometer comparisons under Microsoft Windows, the machines used were Intel LG440GX motherboards with 700 MHZ PIII processors and 2048 MB of RAM. The SCSI disk adapters used were the Adaptec® AIC-7896/7 Ultra2 SCSI host adapters. The machines each had three Quantum ATLAS10K2-TY092L, 9 GB, 10 K disk drives. The operating system used was Windows 2000 Professional with Service Pack 2. The version of Cygwin used to run *pgmeter* was 1.3.2-1 modified and recompiled to support the O_NOBUFFER flag on *open()*.

For the W2K to Linux comparisons, the machines used were Dell® Precision™ 330s, with 1500 MHZ Pentium 4 processors and 512 MB of RAM. The SCSI disk adapters were Adaptec 29160N Ultra 160 SCSI adapters. Each machine had two Fujtisu MAH3364MP, 36 GB, 10K disk drives. The operating systems used were Microsoft Windows 2000 Professional with Service Pack 2 and Linux 2.4.9 as downloaded from www.kernel.org and recompiled for Pentium 4.

Iometer and *Pgmeter* Comparison Cases

Sequential read and write cases were run for record sizes of 4 KB, 8 KB, 16 KB, 32 KB and 64 KB with 1, 4, and 16 outstanding I/O's for each case (30 cases). Random read and write cases were run for record sizes of 1 KB, 2 KB, 4 KB, and 8 KB with 1, 14 and 16 outstanding I/O's for each case (24 cases). The 8 K OLTP, File Server, Web Server, and Workstation workloads were run with 1, 2, 4, 8, 16, and 32 outstanding I/O's (24 cases).

The run rules for the test cases were: 30-second warm-up for each case followed by a 5 minute run to measure the data. Each set of trials was repeated 3 times and the results of the trials were averaged to produce the observed statistic. Since the file-system cache was bypassed during these runs, no reboot of the system between runs was required.

Trademark Information

Linux® is a registered trademark of Linus Torvalds in the United States and other countries.
Microsoft® and Windows® are registered trademarks of Microsoft Corporation.
IBM® is a registered trademark of IBM, Inc. in the United States and other countries.
NetBench® and WinBench® are registered trademarks of Ziff Davis Media Inc.
Intel® and Pentium® are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
Cygwin™ is a trademark of Red Hat, Inc. in the United States and other countries.
SourceForge™ is a trademark of VA Linux Systems, Inc.
SGI™ and XFS™ are trademarks of Silicon Graphics, Inc.
Adaptec® is a trademark or registered trademark of Adaptec, Inc.
Dell® is a registered trademark and Precision™ is a trademark of Dell Computer Corporation.
All other trademarks and copyrights in this paper are property of their respective owners.

Figure 1: W2K NTFS vs Linux 2.4.9 ext2
Sequential Read Benchmark

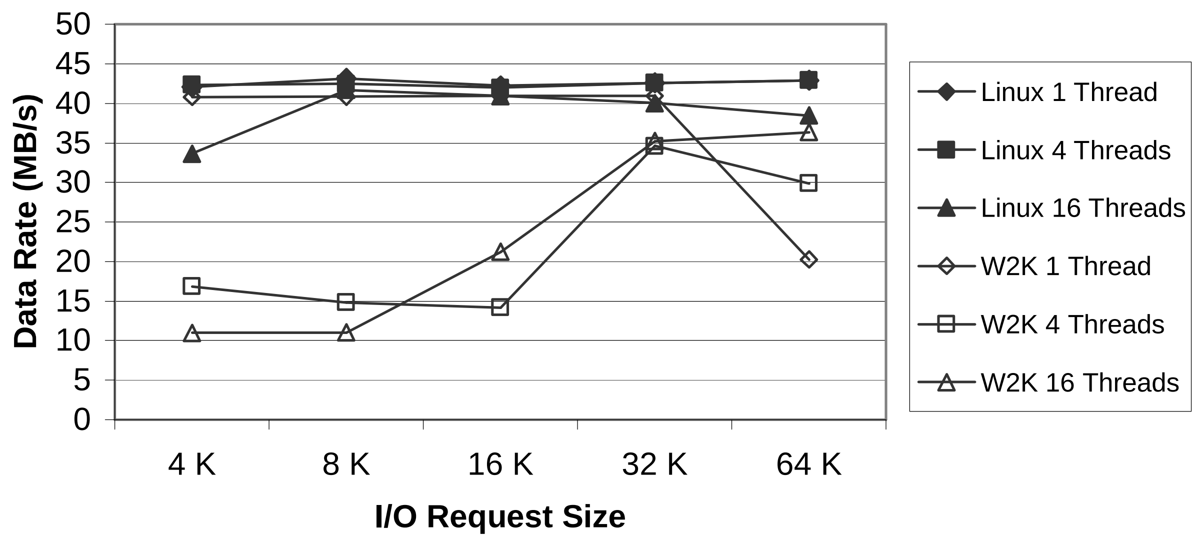
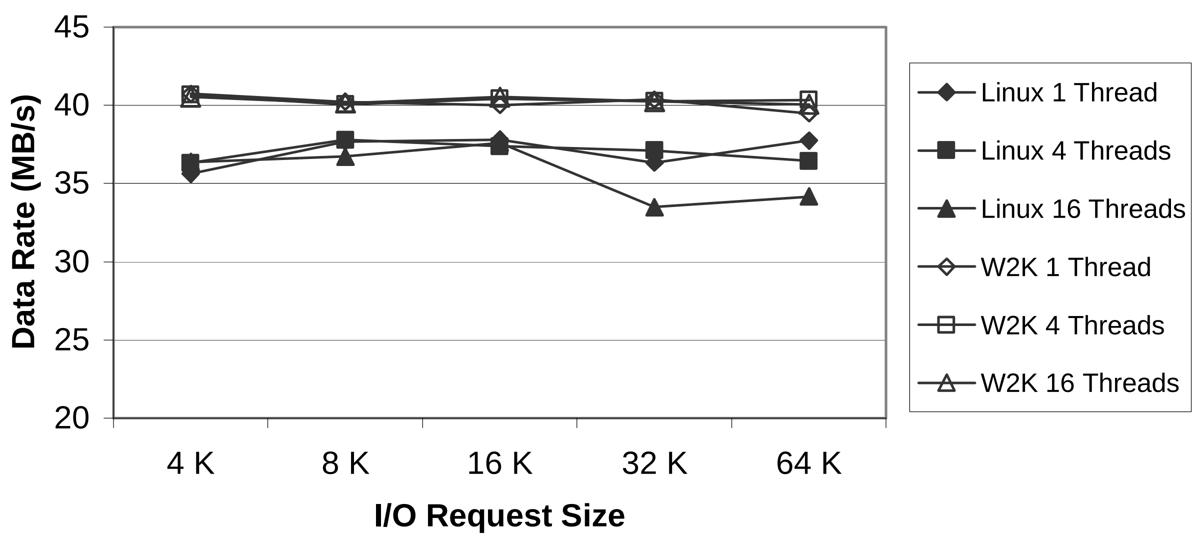


Figure 2: W2K NTFS vs Linux 2.4.9 ext2
Sequential Write Benchmark



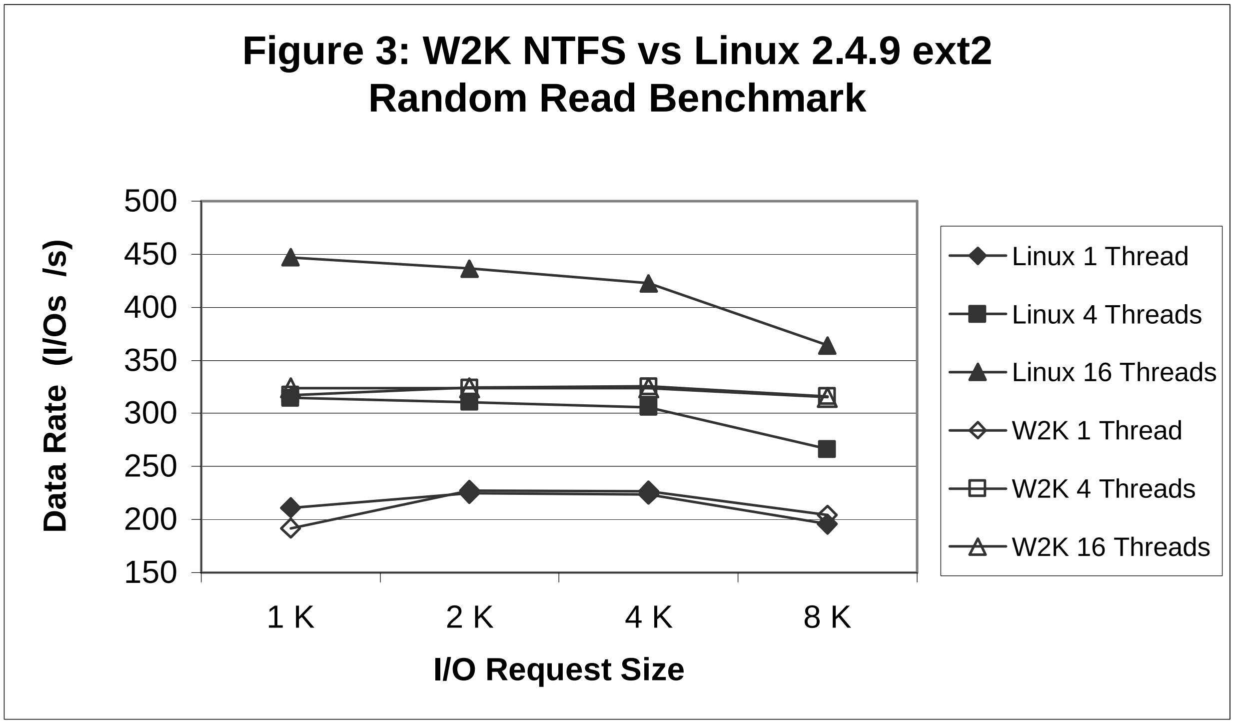


Figure 3: W2K NTFS vs Linux 2.4.9 ext2
Random Read Benchmark
Data Rate (I/Os /s)
500
450
400
350
300
250
200
150
1 K
2 K
4 K
8 K
I/O Request Size
Linux 1 Thread
Linux 4 Threads
Linux 16 Threads
W2K 1 Thread
W2K 4 Threads
W2K 16 Threads

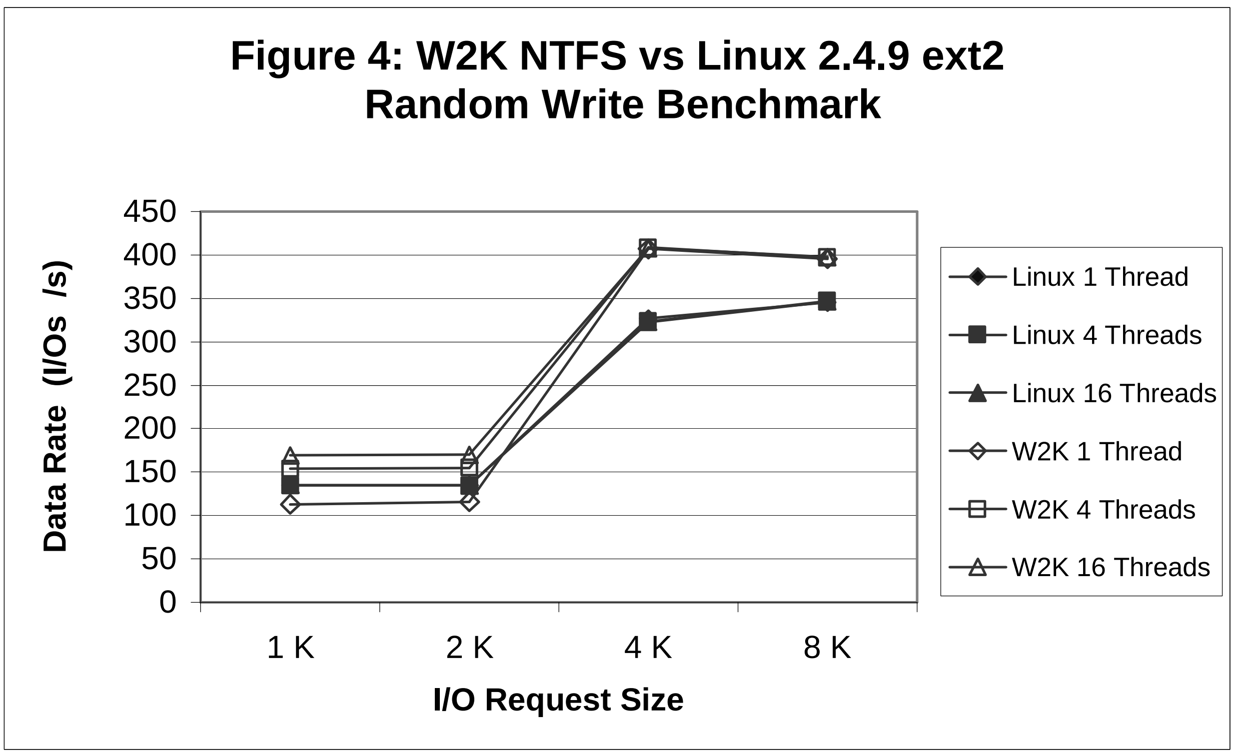


Figure 4: W2K NTFS vs Linux 2.4.9 ext2
Random Write Benchmark
Data Rate (I/Os /s)
450
400
350
300
250
200
150
100
50
0
1 K
2 K
4 K
8 K
I/O Request Size
Linux 1 Thread
Linux 4 Threads
Linux 16 Threads
W2K 1 Thread
W2K 4 Threads
W2K 16 Threads

Figure 5: W2K NTFS vs Linux 2.4.9 ext2 File System Benchmark

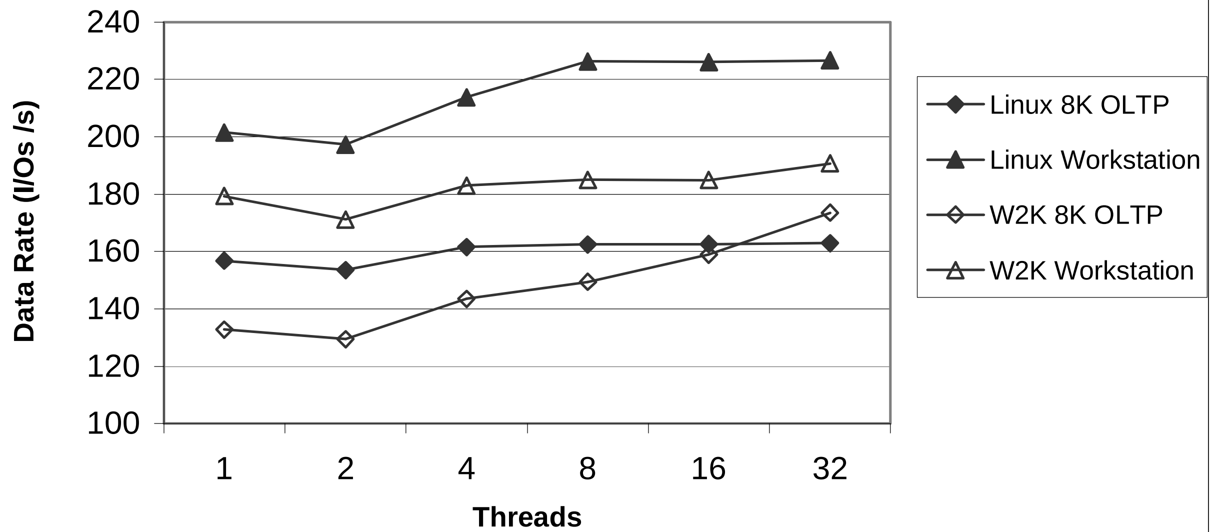


Figure 6: W2K NTFS vs Linux 2.4.9 ext2 File System Benchmark