PostMark: A New File System Benchmark

by Jeffrey Katcher

Abstract

Existing file system benchmarks are deficient in portraying performance in the ephemeral small-file regime used by Internet software, especially:

- electronic mail;
- netnews; and
- web-based commerce.

PostMark is a new benchmark to measure performance for this class of application.

In this paper, PostMark test results are presented and analyzed for both UNIX and Windows NT application servers. Network Appliance Filers (file server appliances) are shown to provide superior performance (via NFS or CIFS) compared to local disk alternatives, especially at higher loads. Such results are consistent with reports from ISPs (Internet Service Providers) who have deployed NetApp filers to support such applications on a large scale.

I. Introduction

Although a panoply of file system benchmarks currently exists, most emphasize raw throughput or performance on large, relatively long-lived groups of files. Current file servers provide services such as electronic mail, netnews, and commerce service, which depend on enormous numbers of relatively short-lived files. These systems are inherited from an era predating the exponential growth of the Internet and were never envisioned as being scaleable to today's required levels.

When planning the acquisition and growth of networked storage systems, it is difficult to predict the performance for Internet duties from existing benchmarks. The only recourse is to measure the performance under the load of electronic mail and netnews services. As using production systems for performance evaluation is both resource intensive and non-deterministic (the load is varied and seldom reproducible), PostMark was created to simulate heavy small-file system loads with a minimal amount of

software and configuration effort and to provide complete reproducibility.

II. The Real World

Analysis of file servers running electronic mail (especially the Simple Mail Transfer Protocol that is nearly universal on the Internet) and netnews shows a strong dependency on an enormous (from over 100,000 into the millions) number of files. Not only are these files relatively small (from one kilobyte to over one hundred kilobytes, depending on content), they are constantly in flux. At any given time, files are being rapidly created, read, written, or deleted, all over the disk drives allocated for these tasks. Small files that continually change are outside the design parameters for most file systems, and perform especially poorly under the UFS file system common to many Unix operating systems.

III. The Benchmark Software

PostMark was designed to create a large pool of continually changing files and to measure the transaction rates for a workload approximating a large Internet electronic mail server.

PostMark generates an initial pool of random text files ranging in size from a configurable low bound to a configurable high bound. This file pool is of configurable size and can be located on any accessible file system.

Once the pool has been created (also producing statistics on continuous small file creation performance), a specified number of transactions occurs. Each transaction consists of a pair of smaller transactions:

- Create file or Delete file
- Read file or Append file

The incidence of each transaction type and its affected files are chosen randomly to minimize the influence of file system caching, file read ahead, and disk level caching and track buffering. This incidence can be tuned by setting either the read or create bias parameters to produce the desired results.

When a file is created, a random initial length is selected, and text from a random pool is appended up to the chosen length. File deletion selects a random file from the list of active files and deletes it.

When a file is to be read, a randomly selected file is opened, and the entire file is read (using a configured block size) into memory. Either buffered or raw library routines may be used, allowing existing software to be approximated if desired.

Appending data to a file opens a random file, seeks to its current end, and writes a random amount of data. This value is chosen to be less than the configured file size high bound. If the file is already at the maximum size, no further data will appended.

When all of the transactions have completed, the remaining active files are all deleted (also producing statistics on continuous file deletion).

On completion of each run, a report is generated showing:

- Elapsed time
- Elapsed time spent performing transactions and average transaction rate (files/second)
- Total number of files created and average creation rate (files/second)

- Number of files created initially and average creation rate (files/second)
- Number of files created during sequence of transactions and average creation rate (files/second)
- Total number of files read and average rate (files/second)
- Total number of files appended and average rate (files/second)
- Total number of files deleted and average deletion rate (files/second)
- Number of files deleted after transactions were complete and average deletion rate (files/second)
- Number of files deleted during sequence of transactions and average deletion rate (files/second)
- Total size of data read and average input rate (bytes/second)
- Total size of data written and average output rate (bytes/second)

A portable random number generator (derived from the Unix reference implementation) is included in the PostMark distribution ensuring identical initial conditions across different platforms and operating systems.

PostMark currently operates under Unix (Sun Microsystems Solaris 1 and Solaris 2, Digital Unix) and Microsoft Win32 environments.

IV. Configuration and Usage

PostMark is operated from a command line interface, even in the Win32 environment. A configuration file (specified at the command line) may contain a list of commands to be run immediately. When run, the program identifies itself and presents its command prompt:

```
PostMark v1.00 : 8/11/97
pm>
```

At the command prompt, a summary of commands is available:

```
pm> ?
set size - Sets low and high bounds of files
set number - Sets number of simultaneous files
set seed - Sets seed for random number generator
set transactions - Sets number of transactions
set location - Sets location of working files
set read - Sets read block size
set write - Sets write block size
set buffering - Sets usage of buffered I/O
set bias read - Sets the chance of choosing read over append
set bias create - Sets the chance of choosing create over delete
run - Runs one iteration of benchmark
show - Displays current configuration
help - Prints out available commands
quit - Exit program
pm>
```

The "show" command displays the current configuration that will be used on the next run. Displayed below is the system default configuration:

```
pm>show
Current configuration is:
Transactions: 500
Files range between 500 bytes and 9.77 kilobytes in size
Random number generator seed is 42
The base number of files is 500
The working directory is: .
Block sizes are: read=512 bytes, write=512 bytes
```

```
Biases are: read/append=5, create/delete=5
Using Unix buffered file I/O
pm>
```

If this configuration is run:

- 500 files will be initially created, between ½ and 10 Kilobytes in size
- The files will be processed in the current directory
- All reads will occur in 512 byte blocks
- All writes will occur in 512 byte blocks
- Reads and appends are equally likely to occur
- Creates and deletes are equally likely to occur
- File operations will be performed using the standard buffered I/O in the runtime library

The "set size" command allows the file range to be specified.

```
pm> set size 10000 20000
```

This example allows files to be created from 10,000 to 20,000 bytes in length. Providing only one number will force all files to be created at the specified length.

The "set number" command sets the number of files to be initially created before the transactions begin. Any integer greater than one is accepted, but the file system must have space for the resulting files.

The "set seed" command sets the initialization value for the random number generator. This should seldom need to be set as the results will then differ from previous runs. Any integer value is acceptable as input.

The "set transactions" command specifies the number of create/delete and read/append transactions that will occur during the next run. Any integer value greater than one is acceptable.

The "set location" command specifies the working directory for the files produced during a program run.

```
pm> set location /usr2/tmp
```

The "set read" and "set write" commands specify how much data is read or written to files at a time. An integer value greater than one is required and should be less than the maximum file size.

The "set buffering" command specifies whether buffered stdio function calls should be used instead of the lower level raw system calls. The valid parameters are "true" and "false" and default to true (buffering is enabled).

The "set bias read" and "set bias create" commands allow the user to select how often an operation occurs while transactions are selected. The allowed values range from -1,0 to 10.

For "set bias read" -1 disables all read and append operations. Otherwise, when the value is lower, append is much more likely to occur. When it is higher, read is more likely. The default value of 5 allows the chances to be equal.

For "set bias create" -1 disables all create and delete operations. Otherwise, when the value is lower, delete is much more likely to occur. When it is higher, create is more likely. The default value of 5 allows the chances to be equal.

The "run" command starts a program run using the current configuration. After the initial files are created, periods are printed to show transaction progress. When ten are printed, all transactions are complete and the remaining files will be deleted. Afterwards, the results will be printed:

```
pm>run
Creating files...Done
Performing transactions..........Done
Deleting files...Done
Time:
        41 seconds total
        17 seconds of transactions (29 per second)

Files:
        740 created (18 per second)
                Creation alone: 500 files (41 per second)
                Mixed with transactions: 240 files (14 per second)
        237 read (13 per second)
        262 appended (15 per second)
        740 deleted (18 per second)
                Deletion alone: 480 files (40 per second)
                Mixed with transactions: 260 files (15 per second)

Data:
        1.31 megabytes read (31.86 kilobytes per second)
        4.20 megabytes written (102.49 kilobytes per second)
pm>
```

Large numbers should be selected for the initial file pool to provide a realistic working set and to prevent caching effects from hiding performance deficiencies. Baseline results were obtained at the 1,000 and 20,000 file levels. The number of transactions should also be large (10,000+) to allow the system to attain a state of equilibrium.

V. The Results

PostMark has been run on a variety of system and storage configurations, both attached and networked. Both Network File System (NFS) and Microsoft CIFS remote storage access protocols were used as necessary.

PostMark was configured in three different ways:

- 1000 initial files and 50000 transactions
- 20000 initial files and 50000 transactions
- 20000 initial files and 100000 transactions

All other PostMark parameters were left constant at the default values.

The NFS benchmark was performed on a Sun Microsystems Ultra 1/170 with 256 Mbytes RAM running under the Solaris 2.5 operating system. The following configurations were tested:

- UFS - The Solaris standard Unix File System (derived from the Berkeley Fast File System)
- TMPFS - A memory-based temporary file system operating in buffer cache (no persistent files, but included to show hardware capabilities without disk bottlenecks)
- ODS/R0 - A software disk array (Sun Online Disk Suite) providing only striping
- ODS/R5 - A software disk array providing parity and striping across the pool of disk drives (RAID5)

- NFS/F330 - NFS to Network Appliance F330 over 100 Mbit/s CDDI (WAFL/RAID4)
- NFS/F630 - NFS to Network Appliance F630 over 100 Mbit/s CDDI (WAFL/RAID4)

1000/50000	**UFS**	**TMPFS**	**ODS/R0**	**ODS/R5**	**NFS/F330**	**NFS/F630**
Transactions per second	36	2000	63	23	139	253
Data read (Kbytes/sec)	115.67	4880	199.73	74.13	441.71	799.91
Data written (Kbytes/sec)	118.27	7330	204.22	75.79	451.64	817.89

Table 1: PostMark Results for Unix and NFS (1,000 initial files and 50,000 transactions)

20000/50000	**UFS**	**TMPFS**	**ODS/R0**	**ODS/R5**	**NFS/F330**	**NFS/F630**
Transactions per second	15	438	29	14	76	176
Data read (Kbytes/sec)	29.93	663.64	56.60	27.05	177.68	383.41
Data written (Kbytes/sec)	54.22	1530	102.54	49.00	321.88	694.58

Table 2: PostMark Results for Unix and NFS (20,000 initial files and 50,000 transactions)

20000/100000	**TMPFS**	**ODS/R0**	**ODS/R5**	**NFS/F330**	**NFS/F630**
Transactions per second	335	30	14	74	169
Data read (Kbytes/sec)	613.03	73.19	35.05	204.72	446.69
Data written (Kbytes/sec)	1160	101.17	48.46	282.98	617.45

Table 3: PostMark Results for Unix and NFS (20,000 initial files and 100,000 transactions)

The results demonstrate that UFS is an extremely poor performer in the PostMark application domain, and degrading even more under heavy load. When software-based disk array solutions are tested (still using UFS as a file system), performance is improved in a striped environment, but RAID5 is worse than the standard UFS configuration. NFS performance with NetApp filers, however, ranges from four times to nearly an order of magnitude better than the baseline configuration, demonstrating the applicability of network attached storage to this class of solution. TMPFS provides the best overall performance, but as it is effectively a RAM-disk solution, it is not a practical alternative for large-scale deployment.

The CIFS benchmark was performed on a Compaq ProLiant 5000 equipped with a 200 MHz Pentium Pro, 128 Mbytes RAM, and the Compaq SMART-2 Disk Array Controller. The system was running the Microsoft Windows NT 4.0 operating system. The following configurations were tested:

- FAT - The basic Microsoft file system inherited from DOS
- NTFS/R0 - Microsoft NTFS on a striped volume managed by the Compaq Disk Array Controller
- NTFS/R5 - NTFS on a parity/striped (RAID5) volume managed by the Compaq Disk Array Controller

- CIFS/F330 - CIFS to Network Appliance F330 over 100 Mbit/s Fast Ethernet (WAFL/RAID4)
- CIFS/F630 - CIFS to Network Appliance F630 over 100 Mbit/s Fast Ethernet (WAFL/RAID4)

1000/50000	**FAT**	**NTFS/R0**	**NTFS/R5**	**CIFS/F330**	**CIFS/F630**
Transactions per second	212	245	144	135	166
Data read (Kbytes/sec)	675.55	776.42	451.63	427.61	522.02
Data written (Kbytes/sec)	690.74	794.18	461.79	437.22	533.75

Table 4: PostMark Results for Windows and CIFS (1,000 initial files and 50,000 transactions)

20000/50000	**NTFS/R0**	**NTFS/R5**	**CIFS/F330**	**CIFS/F630**
Transactions per second	69	37	66	139
Data read (Kbytes/sec)	168.24	90.61	143.83	275.94
Data written (Kbytes/sec)	304.78	164.14	260.55	499.89

Table 5: PostMark Results for Windows and CIFS (20,000 initial files and 50,000 transactions)

20000/100000	**NTFS/R0**	**NTFS/R5**	**CIFS/F330**	**CIFS/F630**
Transactions per second	41	36	68	141
Data read (Kbytes/sec)	179.88	101.14	178.21	350.31
Data written (Kbytes/sec)	248.64	139.81	246.33	484.23

Table 6: PostMark Results for Windows and CIFS (20,000 initial files and 100,000 transactions)

In the Microsoft Windows NT environment, local disk drives show superior performance to network attached storage for smaller numbers of active files. Under heavier loads, both FAT and NTFS volumes (despite the dedicated hardware support provided by the Compaq SMART-2 Disk Array Controller) degrade at a faster rate than even UFS. The NetApp filers, providing network-attached storage via the CIFS protocol, maintain a much more constant level of performance, allowing a superior quality of service even at the heaviest measured workloads.

VI. Future Directions

PostMark generates its workloads by originating as many sequential operations as it can within a specified time limit. While providing useful results from this workload, PostMark could be extended to provide:

- Multiple writers/single reader
- Single writer/multiple reader
- Multiple writers/multiple readers

These additions would allow flexible simulation of many possible disk intensive workloads in addition to

the existing PostMark model.

VII. Summary

The SPEC SFS benchmark has inspired improvements in NFS file servers offered by a wide variety of vendors, and a similar effect is possible with PostMark. Using the PostMark benchmark to highlight performance bottlenecks triggered by ISP workloads, server vendors (including NetApp) can enhance the performance of their products in the ISP context.

The PostMark results presented in this paper confirm the experiences reported by ISPs (Internet Service Providers), where large-scale loads require the greatest possible performance from every file system. At numerous first-tier ISPs worldwide, NetApp filers have been deployed to address these large and growing requirements for maximum performance without compromising reliability.

Information | About NetApp | News | Products | Sales | Support

---