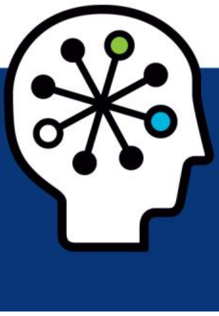
MemFS 2.0 – A Memory-based File System on HP-UX 11iv3



Introduction

A Memory-based File System (MemFS) is a file system that resides in primary memory. It does not normally write data out to stable storage. Thus, MemFS is able to provide extremely high throughput. The purpose of such a file system is to provide fast access for temporary or short lived files that do not need to be kept for an indeterminate time.

Keeping data in memory comes at a cost. It consumes system physical memory. Even with today's large-memory systems, physical memory comes at a premium. A system or application that runs out of available memory at a critical time can cause irreparable loss to the user. As a result, less frequently used data or pages are paged out to a swap device. Under memory pressure, the Virtual Memory (VM) sub-system can de-allocate MemFS pages and reassign them where needed. The MemFS uses system swap device transparently to back the data from the reallocated pages. When needed, this data is brought back into memory.

Implementation Goals

Disk based file system caches file system data in memory and a disk is used as persistent storage. Based on different technologies, disk based file systems require metadata to be updated to the disk either synchronously or asynchronously. Now consider a work load which continuously creates or deletes intermediate file system objects. This kind of work load will generate a stream of metadata update operations to disk. Though journaling can assist in reducing the updates, it still needs metadata to be updated to disk. Due to slow disk transfer rate, performance of such operations reduces.

RAM disk can be alternative for physical disks to overcome slow I/O rates. RAM disk allocates an exclusive chunk of physical memory to host file system data to operate as a device. The I/O to RAM disk is different from I/O to hard disk in a sense that RAM disk driver uses memory to memory copy but hard disk drivers use direct memory access(DMA)for the data transfer. The RAM disk has its own limitations:

- RAM disk requires exclusive allocation of physical memory. Memory is still considered as a sacred resource and file systems already enjoy exclusive reservation of memory in the form of buffer cache. So this exclusive allocation of memory for RAM disk is considered an overhead and this overhead increases by the number of instances of RAM disk based file systems.
- RAM disk uses two memory to memory copies, one from RAM disk to cache and another from cache to user space.
- RAM disk implementation keeps multiple copies of data.
- RAM disk requires a free chunk of contiguous physical memory of the requested size to successfully allocate the RAM disk.

On HP-UX 11iv2, MemFS was developed by reusing the Hierarchical disk based file system (HFS) to treat memory as stream of blocks of specified file system size. As the memory sub-system does not allow the file data pages to use the system swap device, each MemFS instance was associated with a user process which pre-allocates virtual memory. Under system memory/buffer cache pressure, the MemFS file data was swapped into the user process's address space via MemFS pseudo swap driver, which in turn gets swapped on to system swap device by memory management sub-system. As swapping in/out of data between kernel and user space is costly, buffer cache was enhanced to keep data and metadata of MemFS files for longer time than other file systems. Although this file system was substantially faster than any disk based and RAM-disk based file systems, it has several drawbacks, largely due to the limitations of disk based file system layout (HFS) and HP-UX memory subsystem architecture. The drawbacks are as follows:

- Number of files that can be created on a MemFS instance is restricted by file system size rather than the available memory, as HFS dedicates some fixed number of blocks to inodes.

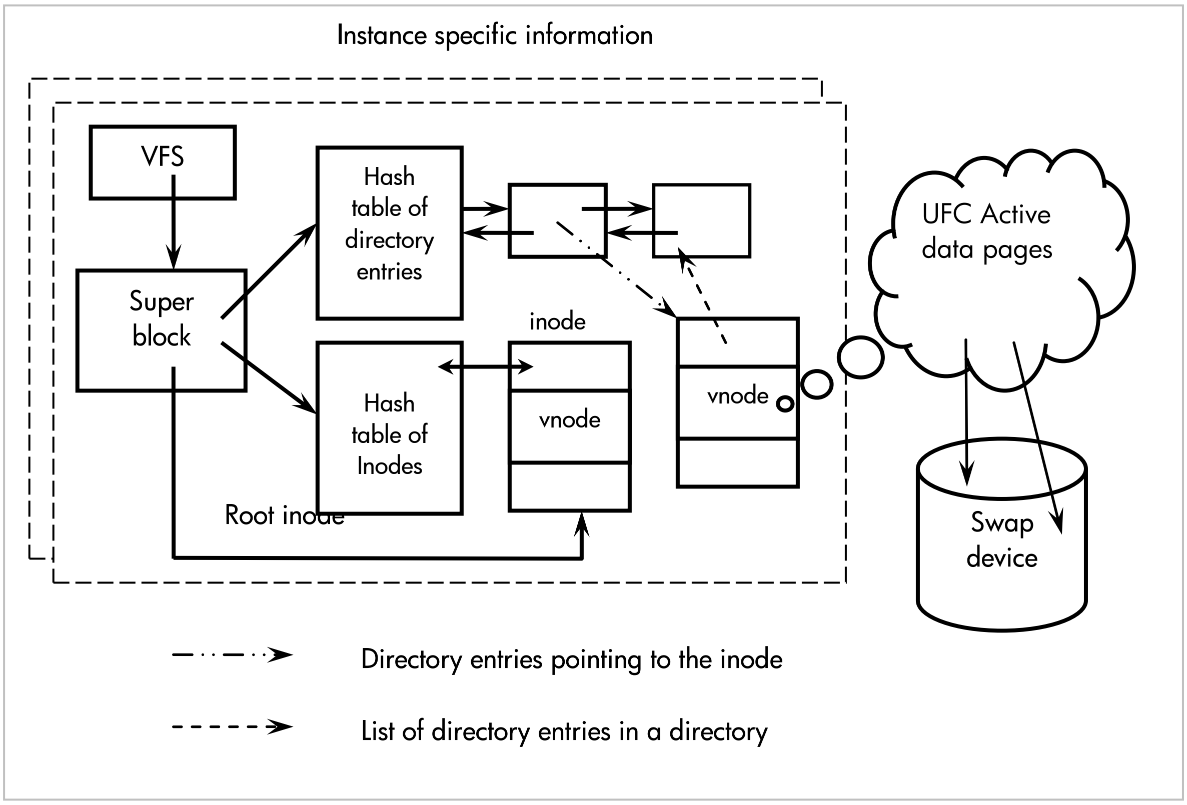
- All the inodes are pre-allocated during the creation of file system, which wastes kernel memory.
- The maximum file system size and file size of MemFS is restricted to 256GB due to the design of inodes in HFS and cannot grow up to the size of available system memory.
- It does not scale well for large directories, as the directory entries are stored and searched sequentially by HFS, which in turn degrades the performance of operations like create, search on large directories.
- Overheads during allocation of blocks for files and directory
- Pre-allocates virtual memory of file system size (similar to RAM disk) during creation of file system, which can cause the system to run out of virtual memory.

A new light weight layout is introduced in MemFS on HP-UX 11iv3 to overcome the above problems, by storing metadata and data in better way to track them faster. The MemFS is entirely implemented in the kernel, without requiring a separate user process. Metadata are stored in the non-page able kernel dynamic memory and file data are kept in the Unified File Cache (UFC). UFC is enhanced to automatically swap the MemFS file data into system swap device under system memory/UFC pressure. The layout just contains minimum necessary information to provide basic functionality of a file system.

Design

The memory file-based system which is available as a configurable product in HP-UX 11iv3 is designed based on the various caches used by the operating system. Figure 1 is an illustration of how MemFS is implemented on HP-UX 11iv3.

Figure 1. MemFS Design


Instance specific information
VFS
Super block
Hash table of directory entries
Hash table of Inodes
inode
vnode
vnode
Root inode
UFC Active data pages
Swap device
Directory entries pointing to the inode
List of directory entries in a directory

The metadata needed are carved from the kernel memory (base cell memory). The data in the files on memory file systems are cached in UFC (Unified file cache) similar to other file systems. But the data are written on to swap devices transparently by UFC during memory pressure.

Data structures

Directory entries

A directory entry, which converts the file name to an inode, is created for each file/directory. Directory entries are maintained in the hash table using the unique tuple. As directory entries of a directory are evenly distributed across the hash table, the lookup of files in the directory is faster and it scales well for large directories. To support sequential reads (readdir()), each directory keeps track of all the directory entries in it using a double linked list and associates a unique offset with each directory entries. The unique directory offsets within a directory are generated using a per-directory specific counter and a per directory specific freelist of available free offsets for re-use, to improve the performance of create operation

Inodes

Inodes keeps track of data and information of files. Inodes are created using the kernel memory and hence the maximum number of inodes on MemFS is restricted by the available kernel memory. To support remote access of files (using Network file system, NFS), they are maintained in a hash table. Directory entries will point to these inodes itself. Thus only one copy of inode is maintained in memory. Vnodes (virtual node) which will provide file system independent access to outside, is embedded in the inodes. All file types are supported, including symbolic links and block and character special device files are supported.

File cache

Unified file cache (UFC) maintains all the pages associated with MemFS. During UFC or system memory pressure, these pages are moved to swap device. It searches the requested data in memory, and informs MemFS to just initiate the I/O to read the pages from swap incase it is not present. UFC maintains the position of the data on swap and transparently handles bringing them to memory.

MemFS Daemon (memfsd)

To improve the performance of some of the operations, a system wide daemon will be running on the systems with MemFS installed. This will make some of the operations asynchronous. Currently this is being used to free the memory and blocks used by the purged files. The files being removed are added to this daemon to free them up asynchronously.

Operations

Mount

A MemFS file system is created using mount operation. It creates an entry in VFS (virtual file system), directory entries and inodes hash tables and root inode. The root inode will be the root of the directory tree.

Directory entries operation

All the basic operations which deal with directory entries like create, remove, hard links and symbolic links are supported on MemFS file system. The directory entries hash table is searched for the specified name in the directory and corresponding action is performed.

read/write operation

MemFS pass on the read/write requests to UFC similar to other file systems. Using the vnode and the offset passed in, UFC finds the pages and bring them in to the memory from the swap in case they are swapped out. Similarly mmap() support is provided.

Usage

MemFS file systems are created by invoking the mount command with ''memfs'' specified as the file system type. Device argument is not used by MemFS as it uses swap device as the back store. It allocates the required metadata for the file system. The metadata related to MemFS is created from kernel memory. The metadata of MemFS is not swappable. It shares the kernel memory with other kernel resources. The memory usage by MemFS can be limited using memfs_metamax(5) tunable. Either the percentage of kernel memory (from 1-30%) can be specified or the actual memory can be specified, 15% will be the default. The total number of files created on MemFS is limited by this tunable. Although it can be restricted using a mount option per file system. ninode=n can be specified to restrict the maximum inodes that can be created on the file system during mount. The file system is free to grow depending on the swap space available. There is no restriction on the file system size (tested till 512GB), but the amount of free space available on MemFS depends on the amount of unallocated swap space at that instant even though the size of file system was specified. The size of a MemFS file system grows to accommodate the data written to it, but there are some inherent tradeoffs for heavy users of MemFS. MemFS data shares resources with the data and stack segments of executing programs. The execution of very large programs can be affected if MemFS file systems are huge in size. Thus the maximum swap space that can be used by MemFS can be restricted using memfs_swapmax_pct(5) tunable. The maximum usage can be restricted from 0 to 80% of the swap space, where 50% will be the default value. This will allow administrator to ensure most programs can execute. Users who expect to run large programs and make extensive use of MemFS should consider enlarging the swap space for the system. In addition, the file system can also be restricted from growing by specifying size option during mount. size=n(KB|MB|GB) will restrict the file system to grow beyond the specified size. This will not reserve the space for that file system; the free space will still depend on the swap space usage at that instant.

These limits can also be modified using "remount" option in addition with the modified values. MemFS file systems are "largefiles" enabled by default, thus they support file size more than 2GB. There is no file size limitation on MemFS (tested till 512GB). ''read only'' mount of MemFS is not supported as it could leave the file system always empty. All file types are supported, including symbolic links and block and character special device files are supported. UNIX file semantics are supported. Access control list (ACL) to restrict the file access is not supported. Quotas that restrict the number of inodes and blocks usage by user or a group is not supported. Multiple MemFS file systems can be mounted on a single system and they share the system memory and UFC. As it provides faster access to temporary files and the data and files on MemFS are not accessible across mounts and reboots, this is suitable for /tmp and /var/tmp.

Performance

A MemFS is expected to perform much better than normal disk file system for smaller file and file system sizes. As the memory occupied by MemFS increases to a level where swapping starts happening, the performance may begin to taper off. If the swapping becomes excessive, performance will decrease. MemFS performance varies depending on usage and machine configurations. The benchmark tests chosen to characterize the performance of MemFS are: Postmark, Connectathon basic tests and SDET.

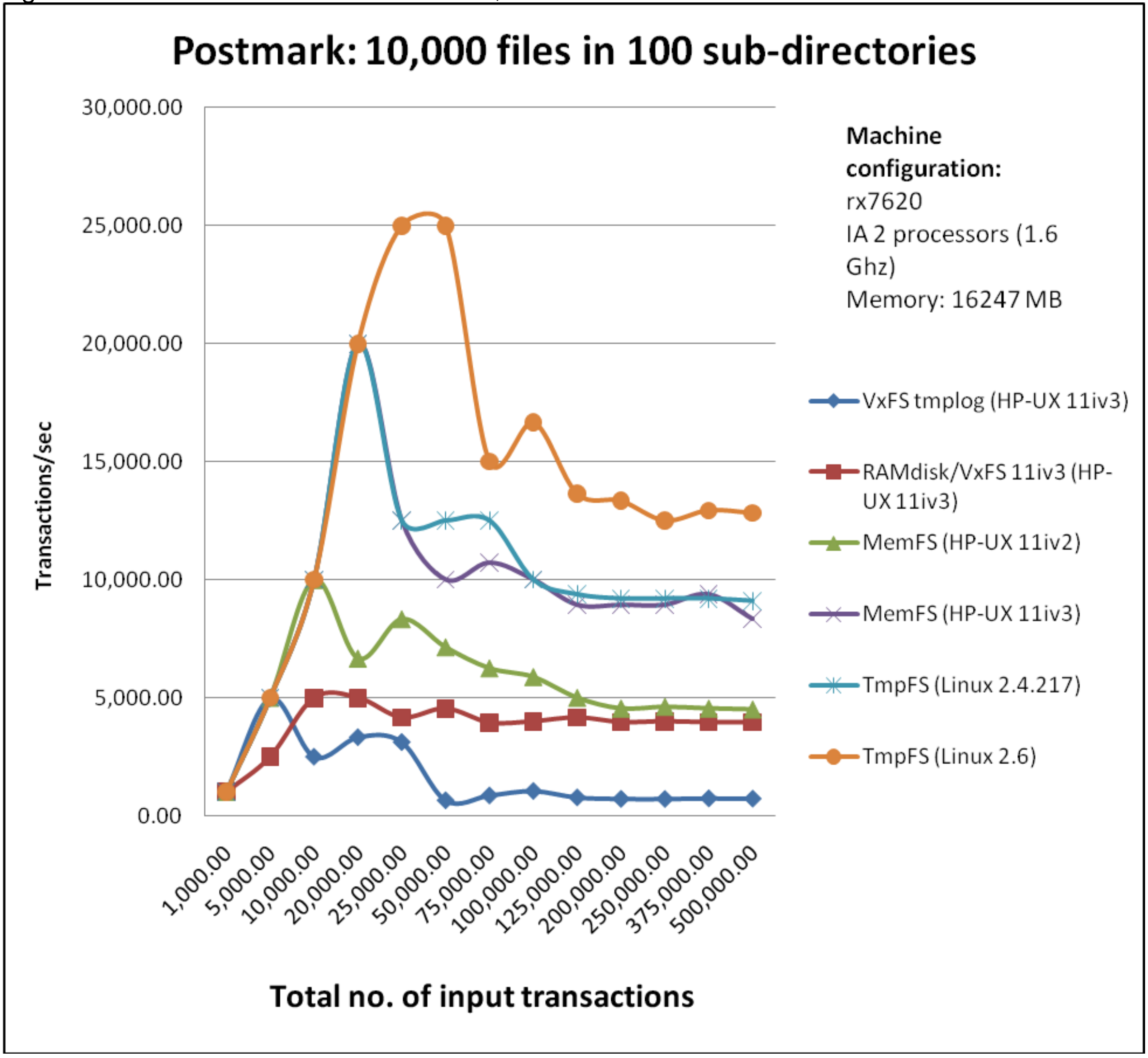
Postmark Benchmark

Network Appliance's Postmark is an appropriate benchmark to use for MemFS as it is an industry-standard benchmark for small file and metadata-intensive workloads. It was designed to emulate Internet applications such as e-mail, netnews, and e-commerce.

Postmark works by creating a configurable sized pool of random text files ranging in size between configurable high and low bounds. These files are continually modified. The building of the file pool allows for the production of statistics on small file creation performance. Transactions are then performed on the pool. Each transaction consists of a pair of create/delete or read/append operations. The use of randomization and the ability to parameterize the proportion of create/delete to read/append operations are used to overcome the effects of caching in various parts of the system.

Test Configuration: The system under test was on HP rx7620 comprising of 2 x 1600MHz Itanium CPUs, 16GB RAM. Postmark v1.5 was used. The following graphs show the effects of applying increasing transaction loads to the various file systems. . Figure 2 is an illustration of the performance of various file systems on various workloads.

Figure 2. Postmark benchmark results for 10,000 simultaneous files and 100 subdirectories

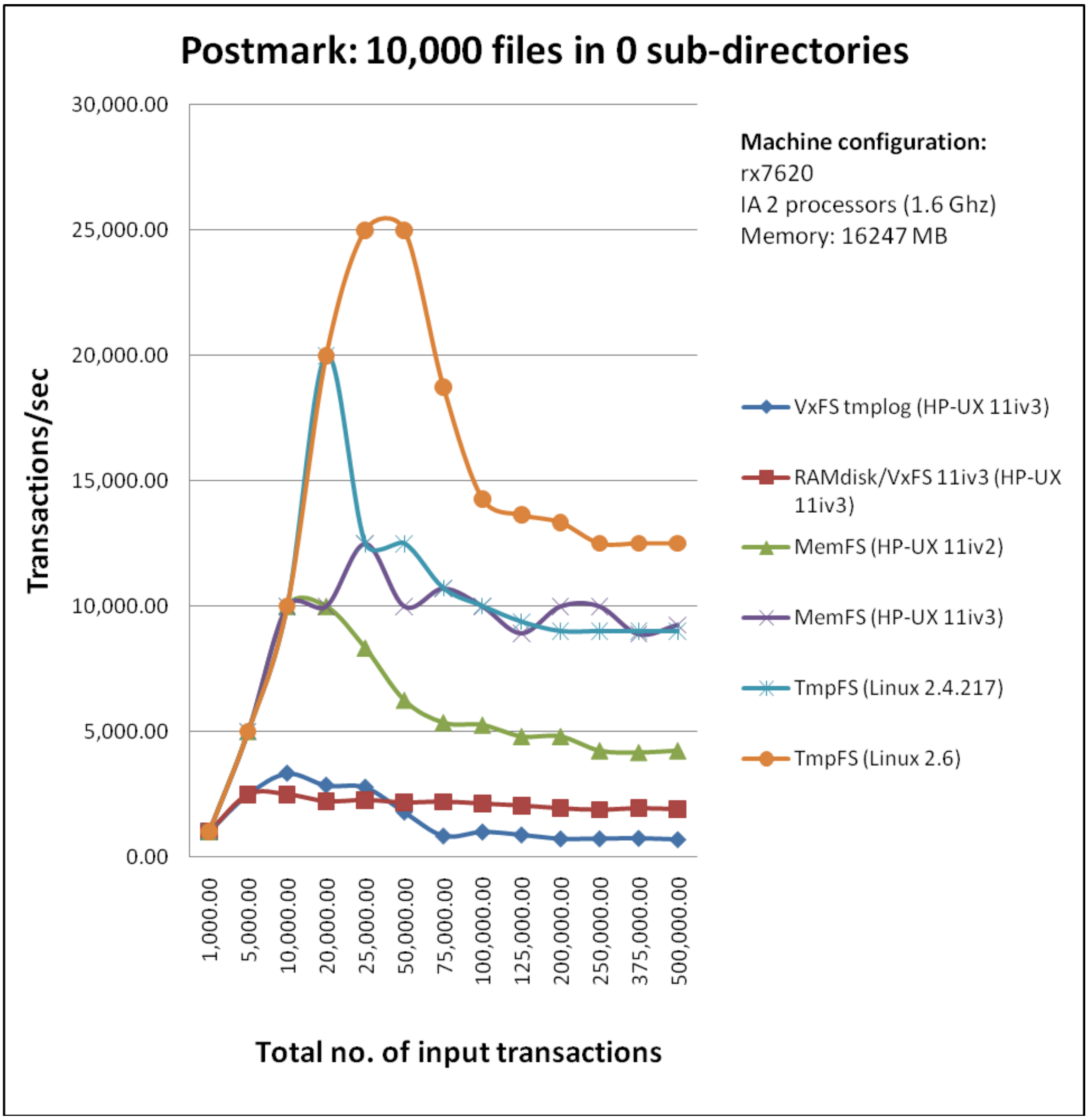


The file systems under test were:

MemFS on HP-UX 11iv3 of size=2GB. memfs_metamax set to 30%.
tmpfs on Red Hat Linux 2.4.217
tmpfs on Red Hat Linux 2.6
MemFS on HP-UX 11iv2 of size=2GB. dbc_max_pct was set to 50% (4GB)
RAMdisk on HP-UX 11iv3: VxFS was the base file system installed on it using the delaylog mount option. VxFS 4.1 was used.

Figure 3 shows Postmark results operating on 10,000 simultaneous files in a single directory. Tmpfs on recent release of Linux, 2.6, shows the best performance, which can be attributed to a light operating system design. But MemFS performs in close proximity to it and better than the tmpfs on previous release of Linux, 2.4.217, MemFS on HP-UX 11iv2 and also RAMdisk.

Figure 3. Postmark benchmark results for 10,000 files and no sub-directories



Connectathon Tests

The Connectathon tests are a collection of micro-benchmarks used to benchmark NFS. The basic tests are used to verify basic operations of “NFS” file system implementations. Figure 3 shows the performance of various file systems on various file operations.

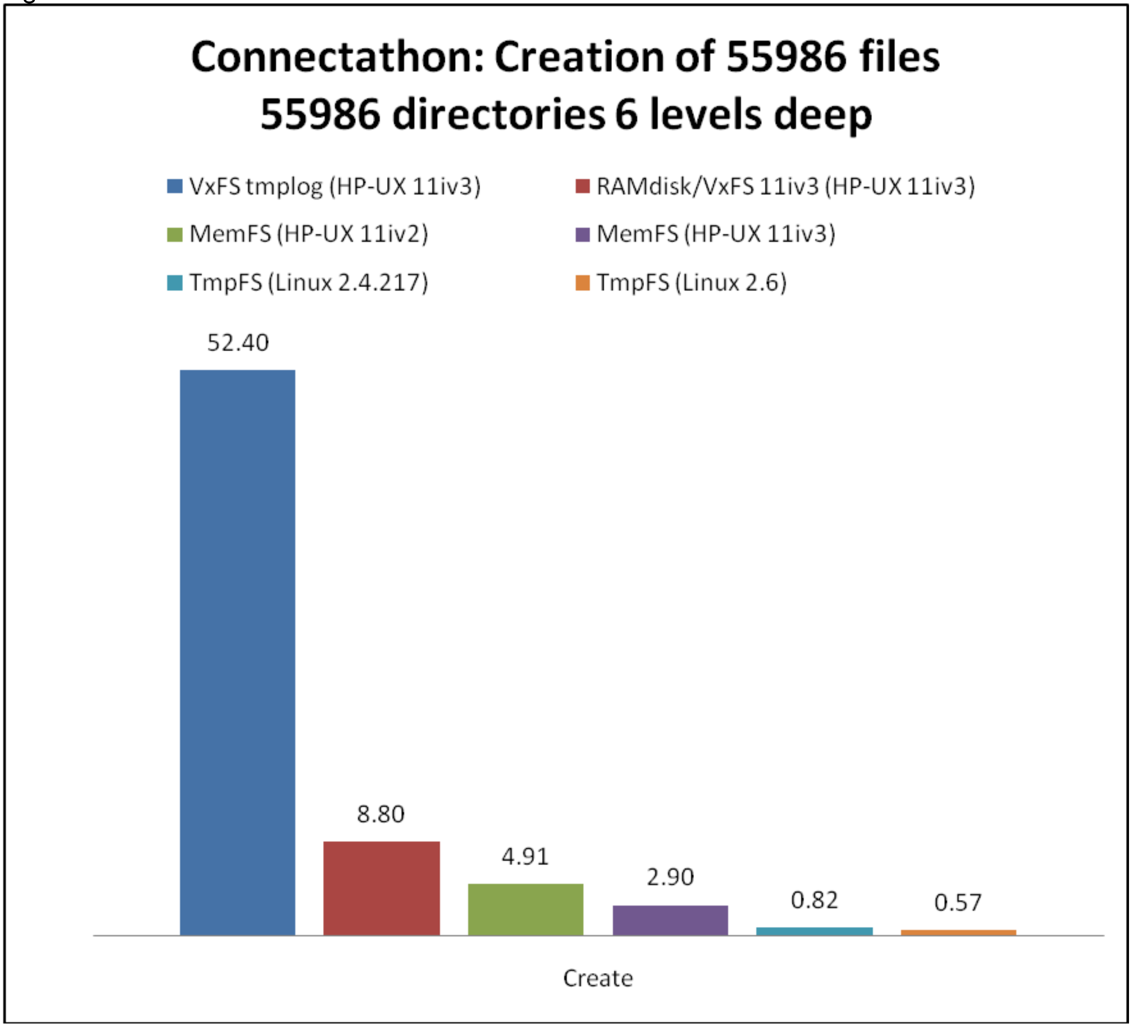
The following tests were used:

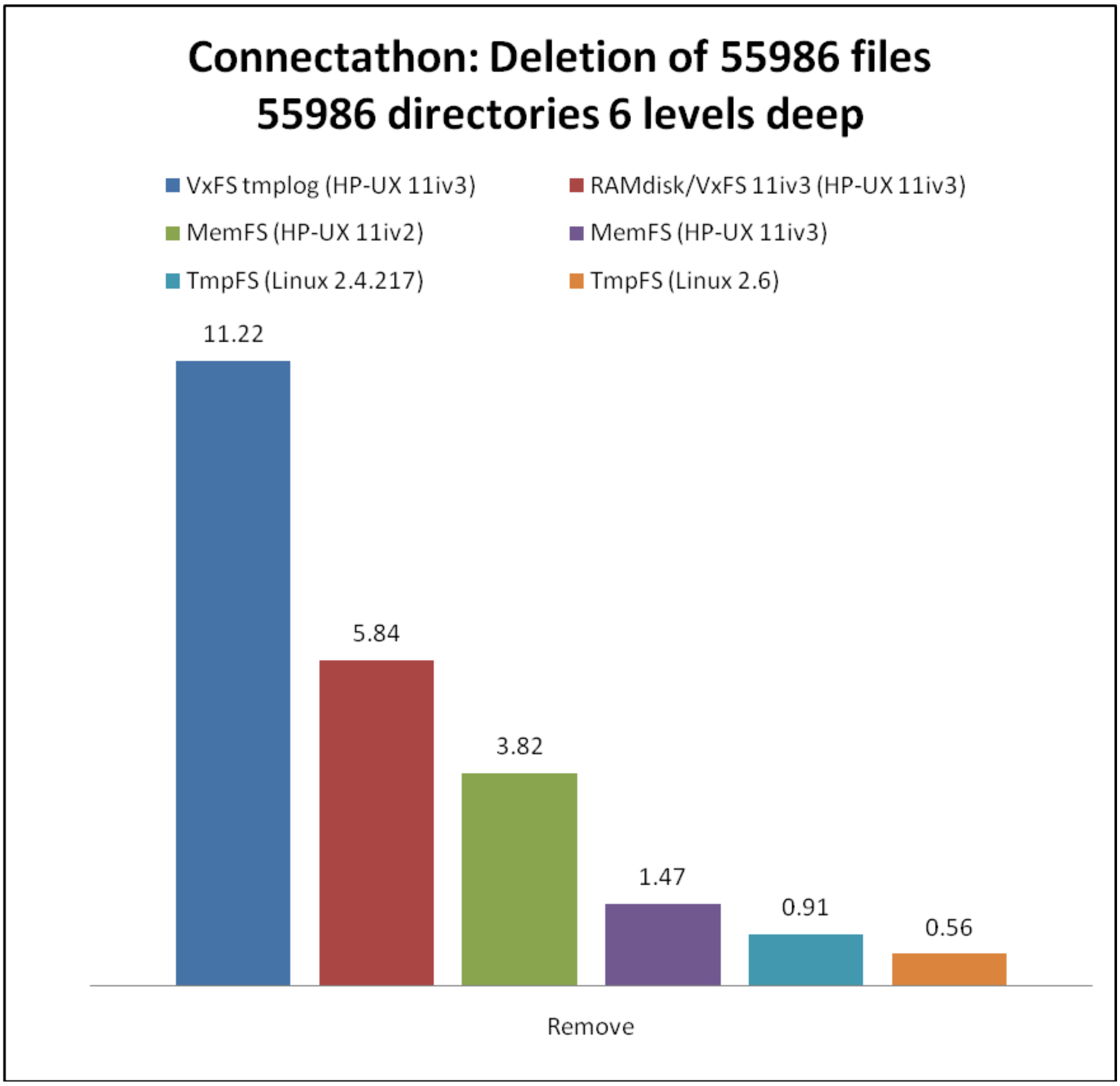
create	Create a directory tree 6 levels deep
remove	Remove the directory tree from create
lookup	stat a directory 500 times
setattr	chmod and stat 10 files 1000 times each
read/write	write and close a 1MB file 10 times, the read the same file 10 times
readdir	read 20500 files in a directory 200 times
link/rename	rename, link and unlink 10 files 200 times

symlink create and read 400 symlinks on 10 files
statfs stat the mount point 1500 times

The system under test consisted of HP rx5670 comprising of 2 x 900MHz Itanium CPUs, 8GB RAM. The Connectathon basic test results indicate that MemFS performs slightly better than RAMdisk/VxFS, and far better than disk based file systems as it takes less time compared to them. It is in close proximity with Linux tmpfs.

Figure 4. Connectathon benchmark results



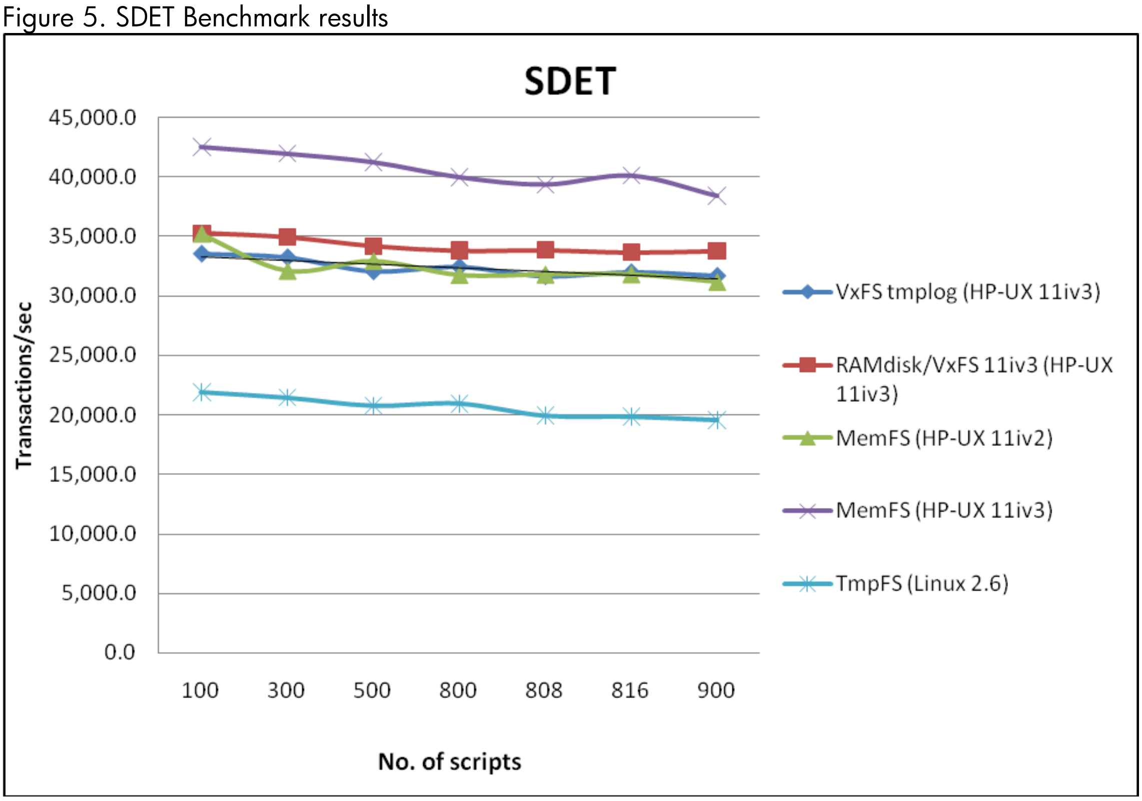


SDET Benchmark

The Software Development Environment Throughput (SDET) benchmark is a highly scalable, multi process benchmark which exercises most of the kernel (except networking). The benchmark was designed to apply a representative model workload to a system at increasing levels of concurrency in order to generate a graph of throughput vs. offered load. SDET is however, a system benchmark.

The system under test consisted of HP rx7620 comprising 8 * 1.6GHz Itanium CPUs and 32GB RAM. Figure 5 shows the performance of the overall system with /tmp being different file systems.

Figure 5. SDET Benchmark results



Configuration Guidelines

A MemFS instance can be created using mount(1m) command,

```
mount [-F memfs]
      [-o [remount]]
      [-o [size=value]]
      [-o [ninode=value]]
      [-o [mode=mode]]
      [-o [user=userID]]
      [-o [group=groupID]]
      [-o suid|nosuid] directory
```

For example, to mount a file system with maximum space 100MB and maximum number of inodes as 1000 on /memfs

```
mount -F memfs -o size=100M,ninode=1000 /memfs
```

To mount a file system with unlimited space and number of inodes, permissions of the file system’s root set to 0777, owner to be user with ID 20, and owning group to be a group with ID 40, on /memfs,

```
mount -F memfs -o mode=0777,user=20,group=40 /memfs
```

To remove the restriction on the file system size on the above file systems,

```
mount -F memfs -o remount,size=0 /memfs
```

As mentioned earlier, the amount of free space on the file system depends on the free swap space at that instant. memfs_swapmax_pct(5) tunable can be used to increase the swap space available for MemFS. 0-80% of the swap space can be specified for MemFS usage. This tunable will just specify the maximum percentage of swap space MemFS can use but this will not reserve the space for MemFS and assures the space for MemFS. In case the MemFS has exhausted the space available for it, ENOSPC will be returned to the application and following message will be displayed on the console.

```
memfs: Cumulative data exceeds memfs_swapmax_pct
```

Similarly the number of inodes that can be created on the file system depends on the free available memory at that instant. ninode just specifies the maximum limit, but does not reserve or assure the number of the files that can be created on that file system. If creation of file or directory fails if this limit has been reached, ENOMEM will be returned to the application. If it fails due to memory deficit, error will be returned and following message will be displayed on the console,

```
memfs: metadata exceeds memfs_metamax
```

memfs_metamax(5) tunable can be used to increase the memory usage of MemFS. Even this will just specify the maximum limit but does not assure memory for MemFS. Maximum of 30% of the memory can be used by MemFS.

Recommended Use

Applications that will benefit most from MemFS are those that perform mostly meta-data intensive operations like creates and deletes of small files and directories. For example: compilers, editors and sorting applications. MemFS never writes out meta-data for files, so directory and file manipulation is always a performance gain. Applications can also interact with MemFS files using the mmap interface. Use MemFS only for temporary files. The /tmp can be mounted as MemFS for better performance.

Limitations and Future Work

MemFS uses static hash table for searching the files on a file system. This could reduce the performance as more number of files is created on the file system. This will be replaced by dynamic growing hash tables in future.

Summary

MemFS shows better performance advantages associated with memory-based file systems and is competitive with other vendors. It also provides significant design advantages over its predecessor and RAMdisk style file systems. MemFS uses memory efficiently because it is not a fixed size, and memory not used by MemFS is available for other uses. MemFS provides additional file system space, and supports UNIX file semantics while remaining fully compatible with other file system types.

5992-4722, March 2009