JFS for Linux

Moving a file system to Linux
February 2002

Steve Best sbest@us.ibm.com
IBM Linux Technology Center

Abstract

This paper describes the features of the Journaled File System (JFS) and some of the history of moving this technology to the Linux operating system.

JFS was designed to support the following requirements:

- Fast recovery from system outages
- Support large files and partitions
- Support a large number of directories & files

JFS provides the following features to meet these requirements:

- Sub-second file system recovery time by journaling metadata only
- 64-bit scalability: PetaBytes range for files and partitions
- B+tree indices used on all file system on-disk structures; B+trees use extensively in place of traditional linear file system structures for performance

What is a File System?

A file system is used to store and manage user data on disk drives; it ensures the integrity of the data written to the disk is identical to the data that is read back. In addition to storing data contained in files, a file system also creates and manages information about the file system itself (i.e. free space, inodes). File system structures are commonly referred to as meta-data. Meta-data is everything concerning a file except the actual data inside the file. Elements of the file such as its physical location and size are tracked by meta-data.

Journaled vs. non-Journaled File System

A Journaling file system provides improved structural consistency and recoverability and faster restart times than non-journaled file systems. Non-journaled file systems are subject to corruption in the event of a system failure, since a logical file operation often takes multiple media I/Os to accomplish and may not be totally reflected on the media at any given point in time. For example, the simple task of writing data to a file can involve numerous steps:

- Allocating blocks to hold the data.
- Updating the block pointers.
- Updating the size of the file.
- Writing the actual data.

If the system is interrupted when these operations are not fully completed, the non-journaled file system ends up in an inconsistent state. In this case, these file systems rely on their fsck utility to examine all of

the file system's meta-data (for example, directories and disk addressing structures) to detect and repair structural integrity problems before restarting. Fsck can be very time consuming, with the amount of time dependent on the size of the partition, the number of directories, and the number of files in each directory. In the case of a large file system, journaling becomes crucial. A journaled file system can restart in less than a second.

JFS uses techniques originally developed for databases to log information about operations on file system meta-data as atomic transactions. In event of a system failure, the file system is restored to a consistent state by replaying the log and applying log records for all of the transactions. The recovery time associated with this log approach is much faster, since the replay utility need only to examine the log records produced by recent file system activity, rather than examine all file system meta-data.

Source of the JFS technology

IBM introduced its UNIX file system as the Journaled File System (JFS) with the initial release of AIX Version 3.1. This file system, now called JFS1 on AIX, has been the premier file system for AIX over the last 10 years and has been installed in millions of customer's AIX systems. In 1995, work began to enhance the file system to be more scalable and to support machines that had more than one processor. Another goal was to have a more portable file system, capable of running on multiple operating systems. Historically, the JFS1 file system is very closely tied to the memory manager of AIX. This design is typical of a closed-source operating system, or a file system supporting only one operating system.

The new Journaled File System, on which the Linux port was based, was first shipped in OS/2 Warp Server for eBusiness in April, 1999, after several years of designing, coding, and testing. It also shipped with OS/2 Warp Client in October, 2000. In parallel to this effort, some of the JFS development team returned to the AIX Operating System Development Group in 1997 and started to move this new JFS source base to the AIX operating system. In May, 2001, a second journaled file system, Enhanced Journaled File System (JFS2), was made available for AIX 5L. In December of 1999, a snapshot of the original OS/2 JFS source was taken and work was begun to port JFS to Linux.

Starting the port

In December 1999 there were three potential journaling file systems just being started or in the process of being developed or ported to Linux. Ext2 was in the process of adding journaling to this file system and it was being called [Ext3]. SGI was in the process of starting to port their [XFS] file system from IRIX. The third file system was being developed by Hans Reiser and it was called [ReiserFS]. None of these file systems were fully functional on Linux in 1999, and IBM believed that JFS was a strong technology and it could add value to the Linux operating system.

Contacts were made with the top Linux file system developers and the possibility of adding yet another journaling file system was explored. One of the basic underlying philosophies of Linux is that choice is good, and the idea of another journaling file system was viewed favorably.

One challenge faced in the open source community is that of community support. One bit of advice the JFS team received from industry contacts during this time was "release early and release often". Community involvement through source code access and contribution is key. IBM started moving JFS to Linux in December, 1999, and by February, 2000, released the first source code. This initial release contained the reference source code, mount/unmount function and support for the ls command on a JFS partition.

Goals for JFS

JFS for Linux had several goals. One, the code was to be released under the GPL license.

The second goal was to complete the port without requiring any kernel changes. By not requiring kernel changes, the view was that it would be more likely to be included in Linux distributions and accepted into the kernel.org source tree.

The third goal is to have JFS run on all architectures supported by Linux. This represents an ongoing community effort. To date, the following architectures are considered tested and supported: ia32, ia64, PowerPC 32 and 64 bit, S/390 31 and 64 bit. In general JFS has no architecture-specific source code. As problems are reported via the community with other architectures, the JFS team works with the community to resolve them.

Summary of JFS Features

The Journaled File System (JFS) provides a log-based, byte-level file system that was developed for transaction-oriented, high performance systems. Scaleable and robust, one of the key features of the file system is logging. JFS, a recoverable file system, ensures that if the system fails during power outage, or system crash, no file system transactions will be left in an inconsistent state. The on-disk layout of JFS will remain consistent without the need to run fsck. JFS provides the extra level of stability with a minimal performance impact when meta-data changes need to be logged.

JFS uses a transaction-based logging technique to implement recoverability. This design ensures a full partition recovery within seconds for even large partition sizes. JFS limits its recovery process to the file system structures to ensure that at the very least the user will never lose a partition because of a corrupted file system. Note that user data is not guaranteed to be fully updated if a system crash has occurred. JFS is not designed to log user data and therefore is able to keep all file operations to an optimal level of performance.

To increase the scalability of the file system JFS uses B+trees pervasively instead of traditional linear file system structures (i.e. ext2 uses a singly-linked list to store the filenames in the directory). B+trees are used for reading and writing extents, which are the most common operations JFS does. B+trees provide a fast search for reading a particular extent of a file. They also provide an efficient way to append or insert an extent in a file.

A general architecture and design overview of JFS is presented in [JFSOverview] paper.

The development of a recoverable file system can be viewed as the next step in file system technology.

Recoverable File Systems

A recoverable file system, such as IBM’s Journaled File System-(JFS), ensures partition consistency by using database-logging techniques originally developed for transaction processing. If the operating system crashes, JFS restores consistency by executing the logredo procedure that accesses information that has been stored in the JFS log file.

JFS incurs some performance costs for the reliability it provides. Every transaction that alters the on-disk layout structures requires that one record be written to the log file for each transaction's sub-operations. For each file operation that requires meta-data changes, JFS starts the logging transaction by calling the TxBegin routine. JFS ends the transaction's sub-operation by calling the TxEnd routine. Both TxBegin and TxEnd will be discussed in more detail in a later section.

Logging

JFS provides file system recoverability by the method of transaction processing technique called **logging**. The sub-operations of any transactions that change meta-data are recorded in a log before they are

committed to the disk. By using this technique, if the system crashes, partially completed transactions can be undone when the system is rebooted. A transaction is defined as an I/O operation that alters the on-disk layout structures of JFS. A completed list of operations that are logged by JFS will be discussed later. One example of an operation that JFS logs is the unlink of a file.

There are two main components of the JFS logging system: the **log** itself and the **transaction manager**. The log is created by the mkfs.jfs format utility and located inside the partition.

There are several JFS data structures that the transaction manager uses during logging and they will be defined next.

Extents, Inodes, Directories

A "file" is allocated in sequences of extents. An **Extent** is a sequence of contiguous aggregate blocks allocated to a JFS object as a unit. An extent is wholly contained within a single aggregate (and therefore a single partition); however, large extents may span multiple allocation groups. JFS uses 24-bit value for the length of an extent. An extent can range in size from 1 to 2(24) -1 blocks. The maximum extent if the block size is 4k would be 4k * 2(24) – 1 bytes and this is equal to (~64G).

Note: this limit only applies to a single extent; in no way limits the overall file size.

Every JFS object is represented by an inode. Inodes contain the expected object-specific information such as time stamps and file type (regular vs. directory, etc.).

JFS dynamically allocates space for disk inodes as required, freeing the space when it is no longer required. This support avoids the traditional approach of reserving a fixed amount of space for disk inodes at file system creation time, thus eliminating the need for users to estimate the maximum number of files and directories that a file system can contain.

JFS has two different directory organizations. The first organization is used for small directories and stores the directory contents within the directory’s inode. This eliminates the need for separate directory block I/O as well as the need for separate storage allocation. Up to eight entries are stored in-line within the inode, excluding the self [.] and parent [..] directory entries. The second organization is used for larger directories and represents each directory as a B+tree keyed on name. This design provides faster directory lookup, insertion, and deletion capabilities when compared to traditional unsorted directory organizations.

A more complete description of JFS’ on-disk layout structures is presented in [JFSLayout] paper.

Transaction Manager

The Transaction Manager provides the core functionality that JFS uses to do logging.

A brief explanation of the transaction flow follows:

A call to TxBegin allocates a transaction "block", tblk, which represents the entire transaction.

When meta-data pages are created, modified, or deleted, transaction "locks", tlck's, are allocated and attached to the tblk. There is a 1 to 1 correspondence between a tlck and a meta-data buffer (excepting that extra tlcks may be allocated if the original overflows). The buffers are marked 'nohomeok' to indicate that they shouldn't be written to disk yet.

In txCommit (), the tlck's are processed and log records are written (at least to the buffer). The affected inodes are "written" to the inode extent buffer (they are maintained in separate memory from the extent buffer). Then a commit record is written.

After the commit record has actually been written (I/O complete), the block map and inode map are updated as needed, and the tlck'ed meta-data pages are marked 'homeok'. Then the tlcks are released. Finally the tblk is released.

Example of creating a file

A brief explanation of the create transaction flow follows:

```
TxBegin(dip->i_ipmnt, &tid, 0);

tblk = &TxBlock[tid];

tblk->xflag |= COMMIT_CREATE;

tblk->ip = ip;

/* Work is done to create file */

rc = txCommit(tid, 2, &iplist[0], 0);

TxEnd(tid);
```

File System operations logged by JFS

The following list of file system operations that change meta-data of the file system so they must be logged.

- File creation (create)
- Linking (link)
- Making directory (mkdir)
- Making node (mknod)
- Removing file (unlink)
- Rename (rename)
- Removing directory (rmdir)
- Symbolic link (symlink)
- Truncating regular file

Log maximum size

The format utility mkfs.jfs creates the log size by using the following formula: which is based on the partition size.

The default size of the log is 0.4 % of the aggregate size and this value is rounded up to a megabyte boundary. The maximum size that the log can be is 32M. The log size is then converted into aggregate blocks.

For example the size of the log file for 15G partition using the default is 8192 aggregate blocks using 4k block size which is equal to 32M. During the design of JFS it was determined that having a log size greater than 32 M wasn’t beneficial.

The log is currently created inside the partition and we are working on a new feature that would allow the log to be external to the partition. By having the log external, the performance of the file system will be increased.

Logredo operations

Logredo is the JFS utility that replays the log file upon start-up of the file system. The job of logredo is to replay all of the transactions committed since the most recent synch point.

The log replay is accomplished in one pass over the log, reading backwards from log end to the first synch point record encountered. This means that the log entries are read and processed in Last-In-First-Out (LIFO) order. In other words, the records logged latest in time are the first records processed during log replay.

A more complete description of JFS' logging is presented in [JFSLog] paper.

Downloading and installing JFS

JFS was included in Alan Cox's 2.4.18pre9-ac4 kernel on 2/14/2002. Alan's patches for 2.4.x series are available from www.kernel.org. You can also down load the kernel source tree and add the JFS patches to this tree. JFS comes as a patch for several of the 2.4.x kernel, so first of all, get the latest kernel from www.kernel.org. At the time of writing this article the latest kernel is 2.4.17 kernel and this kernel will be used in the section below. The JFS patch is available http://oss.software.ibm.com/jfs. The latest release of JFS when this article was written is 1.0.15. You will need both the utilities (jfsutils-1.0.15.tar.gz) and the file system (jfs-2.4-1.0.15-patch.tar.gz) patches. There are several Linux distributions already shipping JFS, Turbolinux, Mandrake, SuSE, so you might not need to doing the steps below to use JFS on your system (see the section below entitled Distributions shipping JFS) to determine if JFS support has been included .

- Down load the kernel (linux-2.4.17.tar.gz)
- place downloaded linux kernel archive in /usr/src dir
- mv linux linux-save
- expand the kernel source tree by using:
- tar zxvf linux-2.4.17.tar.gz
- make dir for jfs source /usr/src/jfs1015
- place jfs patches into /usr/src/jfs1015 dir
- expand the JFS kernel patch by using:
- tar zxvf jfs-2.4-1.0.15-patch.tar.gz
- see the README file in jfs-2.4-1.0.15-patch.tar.gz for additional information
- change to the directory of your kernel source tree
- cd /usr/src/linux
- apply the JFS kernel patches
- patch –p1 < /usr/src/jfs1015/jfs-2.4-common-1.0.15.patch
- patch –p1 < /usr/src/jfs1015/jfs-2.4.17-1.0.15-patch

Configuring the kernel and JFS

To configure the kernel, I use the "make xconfig" command. Now, configure JFS by going to the File systems section of the configuration menu. After configuring the rest of the kernel, exit and save your changes.

Compiling and installing

If you configured JFS as a module, you can just recompile and reinstall your kernel modules by typing" make modules; make install_modules" Otherwise, you need to recompile your kernel by typing make dep; make clean; make bzImage; make modules; make modules_install; make install". Update lilo.conf with the new kernel if you need to and run lilo to make the system aware of the new kernel. Reboot with the new kernel that has JFS included.

Compiling and installing JFS utilities

After the kernel is complied and installed you should make and install the JFS utilities:

- place the jfsutils-1.0.15.tar.gz file into the /usr/src/jfs1015 sub dir
- expand the jfs utilities by using:
- tar zxvf jfs-1.0.15.tar.gz
- cd jfsutils-1.0.15
- see the README in the jfsutils-1.0.15.tar.gz for additional information
- ./configure
- make
- make install

So in the above steps you have built and installed the utilities.

Creating a new partition

To use JFS, you'll need a spare partition. If there is unpartitioned space on our disk, you can create a partition using **fdisk.** My system has space on /dev/hdb so I've create a partition /dev/hdb3 which will be used in the next step to format it as a JFS partition. Reboot your system to make sure that the new partition is available to create a JFS file system on it.

Creating the file system

- mkfs.jfs /dev/hdb3

Mounting the file system

After the file system has been created. It is time to mount it. You will need a mount point and you can do this by creating a new empty directory such as /mnt/jfs, to mount the file system use the following mount command.

- mount –t jfs /dev/hdb3 /mnt/jfs

After the file system is mount you ready to try out JFS. Have fun!

Unmounting the file system

To unmount the jfs system use the umount command with the mount point

- umount /mnt/jfs

Distributions shipping JFS

- Turbolinux 7.0 Workstation (8/2001)
- Mandrake Linux 8.1 (9/2001)
- SuSE Linux 7.3 Intel (10/2001)
- SuSE Linux 7.3 PowerPC (11/2001)
- Turbolinux 7.0 Server(12/2001)
- SuSE Linux Enterprise Server 7 for IBM eServer iSeries and pSeries (1/2002)

Distributions in process of shipping JFS

- Debian (Woody release)

Summary

If you would like to help out on JFS, please join our mailing list or send me e-mail at sbest@us.ibm.com and I'll match your interests to one of the work items that need to be done. Or if you need a journaling file system for your production system try JFS and let me know how it works out for you.

Acknowledgements

References

[JFSOverview]: "JFS overview" Best.,
http://www-4.ibm.com/software/developer/library/jfs.html

[JFSLayout]: "JFS layout" Best, et al.,
http://www-4.ibm.com/software/developer/library/jfslayout/index.html

[JFSLog]: "How the Journaled File System does Logging"Best,
http://www.usenix.org/publications/library/proceedings/als2000/best.html

[XFS]: SGI's XFS file system for Linux. The XFS web page is at
http://oss.sgi.com/projects/xfs.

[ReiserFS]: Namesys' ReiserFS file system for Linux. The ReiserFS web page is at
http://www.namesys.com. ReiserFS was first included in 2.4.1 release of the Linux kernel.

[Ext3]: Developed by Stephen Tweedie, ext3 adds journaling to ext2.
Information for ext3 is available at
ftp://ftp.us.kernel.org/pub/linux/kernel/people/sct/ext3/
Andrew Morton web page
http://www.zipworld.com.au/~akpm/linux/ext3/
Ext3 was first included in 2.4.15 release of the Linux kernel.

[JFS Root Boot HOWTO]
http://oss.software.ibm.com/developer/opensource/jfs/project/pub/jfsroot.html

Source code for JFS is available from
http://www.oss.software.ibm.com/jfs

Trademark and Copyright Information