JFS (64bit FS by IBM)

- **Journaled File System (JFS) provides**
  - a **log-based**,
  - **byte-level file system** that **was developed** for **transaction-oriented**, **high performance systems**.
- **Scalable** and **robust**, **its advantage** over **non-journaled file systems** is its **quick restart capability**:
  - **JFS can restore FS** to a **consistent state** in a **matter of seconds** or **minutes**.
- **While tailored primarily** for the **high throughput** and **reliability requirements** of **servers**
- (from **single processor systems** to **advanced multi-processor** and **clustered systems**),
- **JFS** is **also applicable to client configurations where performance** and **reliability are desired**

Journaling under JFS

- **JFS only logs operations** on **meta-data**,
- **It does not log file data** or **recover** this **data to consistent state**.

- **JFS implements**:
  - **synchronous write to** the **log**
  - **group commit**
    - **which combines multiple synchronous write operations**
    - **into** a **single write**
  - **asynchronous write to** the **log**

Extent-based addressing structures

- **JFS uses**
  - **extent-based addressing structures**,
  - **along** with **aggressive block allocation policies,**
  - **to produce compact**, **efficient**, and **scalable structures for mapping logical offsets within files to physical addresses on disk**
- An **extent is** a **sequence** of **contiguous blocks allocated to** a **file** as a **unit and is described by** a **triple**, **consisting of**
  - <**logical offset, length, physical**>
- The **addressing structure is** a **B+ tree of extents**
  - **populated with extent descriptors** (the triples above),
  - **rooted in** the **inode and**
  - **keyed by logical offset within** the **file**

block sizes

- **JFS supports block sizes** of
  - **512, 1024, 2048, and 4096 bytes** on a per-file system basis,
  - **allowing users to optimize space utilization** based on their application environment.
- **Smaller block sizes** reduce the **amount of internal fragmentation within files and directories** and **are more space efficient**.
- However, **small blocks can increase path length since**
  - **block allocation activities may occur**
  - **more often** than if a large block size were used.
- The **default block size is 4096 bytes**
  - **since performance**,
  - **rather than space utilization**,
  - is **generally the primary consideration** for **server systems**.

Dynamic disk inode allocation

- **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** the **file system creation time**,
- **thus eliminating the need for users to estimate the maximum number of files and directories that a file system will contain**.

- **Additionally**, this **support decouples disk inodes** from **fixed disk locations.**

Directory organization

- **2 different directory organizations are provided**.
- 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 to allocate separate storage.
  - **Up to 8 entries may be stored in-line within the inode**, excluding the self(.) and parent(..) directory entries, which are stored in separate areas of the inode.
- The **second organization** is used for **larger directories** and **represents each directory** as a **B+tree keyed on name**.
- **It provides faster directory lookup**, insertion, and deletion capabilities when compared to traditional unsorted directory organizations.

Sparse and dense files

- **JFS supports both sparse and dense files**, on a **per-file system basis**
- **Sparse files** allow data to be written to random locations within a file without instantiating previously unwritten intervening file blocks.
- The **file size reported** is the **highest byte that has been written to**, but the **actual allocation of any given block in** the **file does not occur until** a **write operation is performed on that block**.

- **For example**, suppose a **new file** is **created in a file system** designated for sparse files. An application writes a block of data to **block 100** in the file.
- JFS will report the size of this file as 100 blocks, although only 1 block of **disk space has been allocated to it**. **If the application next reads block 50 of** the **file**, JFS will return a block of zero-filled bytes. Suppose the application then writes a block of data to block 50 of the file.
- JFS will still report the size of this file as 100 blocks, and now 2 blocks of disk **space have been allocated to it.**
- **Sparse files are of interest** to applications that require a **large logical space but only use a (small) subset of this space.**

Sparse and dense files

- For **dense files**,
- **disk resources are allocated to cover the file size**.
- In the above example,
  - the **first write** (a block of data to block 100 in the file)
  - **would cause 100 blocks of disk space to be allocated to the file.**

- A **read operation** on **any block that has been implicitly written**
  - to **will return a block of zero-filled bytes**,
  - **just as in the case of** the **sparse file**.

Internal JFS (potential) limits

- **JFS is a full 64-bit file system. All of the appropriate file system structure fields are 64-bits in size.**
- This **allows JFS to support both large files and partitions**.

- **File system size**

- The **minimum file system size** supported by JFS is **16 Mbytes**.

- The **maximum file system size** is a **function of the file system block size and the maximum number of blocks supported** by the file system meta-data structures.

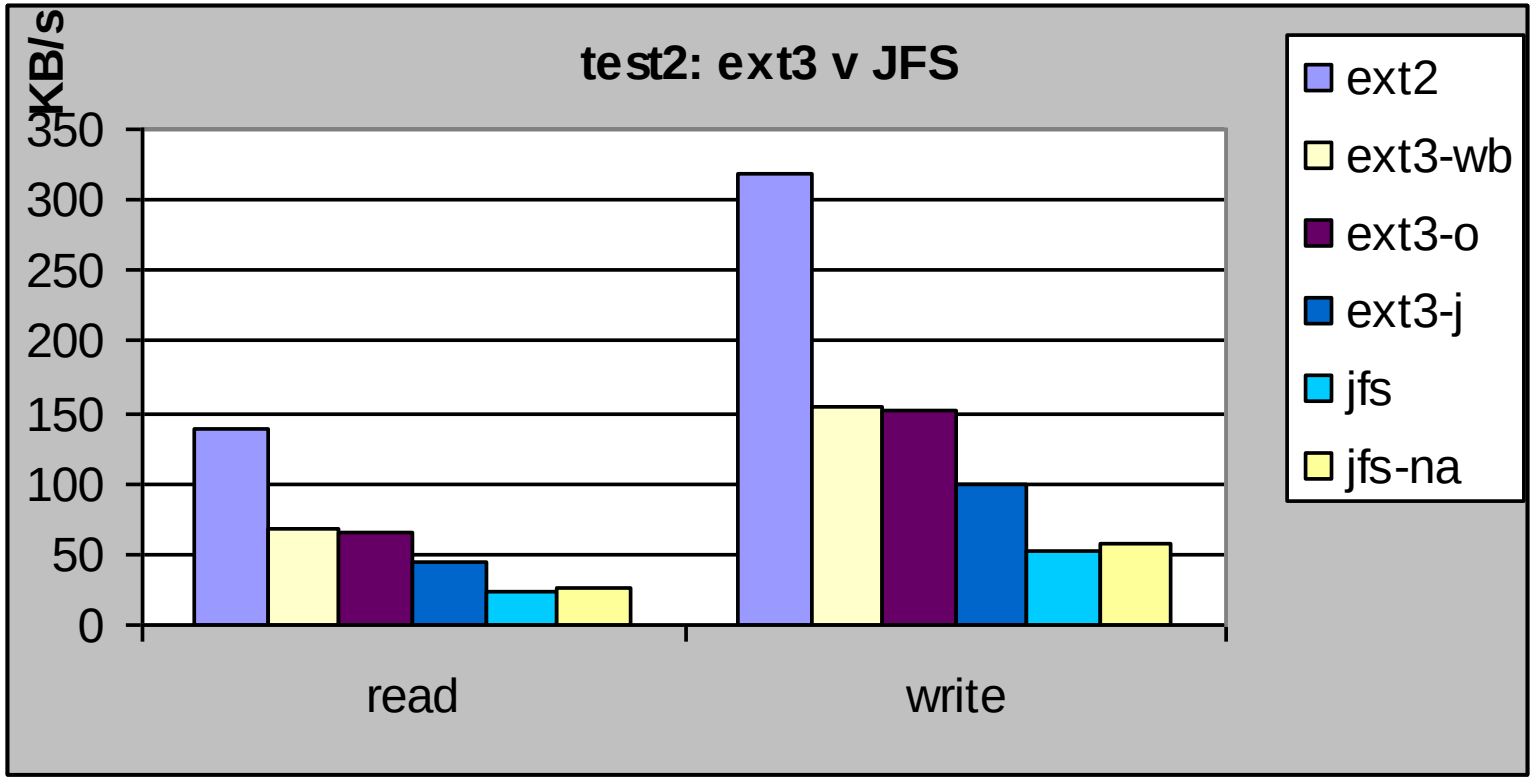
- **JFS will support a maximum FS size of**
  - **512 terabytes** (with block size512 bytes)
  - to
  - **4 petabytes** (with block size 4 Kbytes).

Internal JFS (potential) limits

- **File size**
  - The **maximum file size is the largest file size** that **virtual file system framework supports.**
  - For example, if the **frame work only supports 32-bits**, then **this limits** the **file size**.

- **Removable media**
- **JFS will not support diskettes** as an underlying file system device.

Ext3 v JFS


test2: ext3 v JFS
KB/s
350
300
250
200
150
100
50
0
read
write
ext2
ext3-wb
ext3-o
ext3-j
jfs
jfs-na