CMPSCI 377 Operating Systems **Fall 2005**

Lecture 22: December 8

Lecturer: Emery Berger *Scribes: Billy Dean and Zeid Rusan*

Last time: Distributed Computing

Today we're going to learn about Distributed File Systems:

- Remote File Access
- Remote Caching
- Network File Systems
- Servers with state and without replication

22.1 Distributed File Systems

22.1.1 Remote File Access

When files are accessed remotely, first the local system calls to get the remote file have to be changed into RPCs (Remote Procedure Calls).

Remote Caching on a Local Disk:

- + Reduces access time
- + Safe if a node fails
- - Hard to keep consistent local copy

Remote Caching on Local Memory:

- + Quick access time
- + Works without a disk
- - Hard to keep consistent copy in memory
- - Power must be constant or data will be lost

22.2 Cache update policies

Write-through

- + Reliable

- - Low performance

Write-back:

- + Quick
- + Reduces network traffic
- - Users machine crashes, causing data loss

Cache consistency:

Client initiated consistency checks the consistency at every file access or at given intervals.
Server initiated consistency checks the consistency on a timer

- Server detects conflicts and invalidates the cache
- Server needs to know which clients have cached which parts of the files
- Server also needs to know which clients are readers and which are writers

22.3 Network File Systems

Defines a set of Remote Procedure Call operations for remote access to files

1. Directory search, read directory entries
2. Manipulating links and directories
3. Accessing file attributes
4. Read/Write files

NFS changes all requests into RPCs

- + NFS is good because it doesnt rely on all nodes being the same
- - However it is not consistent
- - Everything is done remotely, therefore...
- - It is 10 to 100 times slower than local procedure calls
- - Caching is bad as well because its all done through RPC. E.g. Four round trips of RPC are required for creating a new file.

Tip: In Linux, the /tmp directory is mounted locally. So you can copy files you want to compile into there first to get faster compiles.

Sun's NFS:

- The standard for UNIX

- Intended for LANs
- Not designed to run on the Internet's scale
- Nodes can be clients and servers.
- Windows uses Somba/SombaFS.

22.4 Advanced File Systems

- Motivation and Review
- Journaling
- Log structured file systems

Servers have special needs (that the default FS "ext2" doesnt support well):

- Large hard-disk partitions
- Quick crash recovery (takes hours in ext2)
- High-Performance I/O (Does not scale well with the number of files)
- Storing thousands of files, terabytes of data

22.4.1 Blocks and Fragmentation

- Logical Block - smallest unit of storage allocated by the FS
- Internal fragmentation - happens when a file does not fill block completely. Ex. file=10kb, block=8kb, wastes 6k.
- External fragmentation - happens when the logical blocks of a file are scattered. This causes poor performance.

22.4.2 Review: Organization

- Extent - maps for where files are stored
- Contiguous blocks - Triple(fileOffSet, StartingBlockNumber, Length)
- File offset - consists of an offset of extents first block for file start
- Start block number - First block in extent
- Length - Number of blocks in extent
- Inode - Stores information about file. e.g. permissions, types of links, direct and indirect pointers, etc
- Metadata - "A-Time" of file; last time it was accessed

Metadata updates:

- Lots being updated
- Scattered on disk; slow and non-atomic.

Note: Making Inodes bigger or adding more indirection will give you bigger file capacity, but indirection requires too much hopping.

22.4.3 Repairing File System Inconsistency

Unix's command /fsck is a FS check: It detects and repairs structural problems, etc by marching though the entire disk looking for faults in all the Metadata. It is run after power outages etc But...:

1. - It is SLOW
2. - It might not work

The main problem: We have non-atomic writes which then have to recover.

A solution:

22.4.4 Journaling File Systems

- Automatically write all planned transactions into a log.
- Recovery in JFS: Some updates are fully committed to file system, find journal entries, replay the actions.

Journaling is great for Metadata.

1. + Fast
2. + Guarantees consistency
3. - Doesn't guarantee zero data loss

22.4.5 Log-structured File System

- + Preserves data integrity
- + Performance

Schedules periodic compaction, places Inode map to keep track of all locations. You roll back to the last Inode map if something happens.

Examples: "Sprite" LFS, ReiserFS and X3 File systems: Data is safe and it is faster:

- + Outperforms UNIX FS for small writes and matches it for reads and large writes
- + Utilizes 70 percent of disk bandwidth even with the overhead of segment cleaning included

LFS structure:

- Inode maps maintain the location of data, directories, etc.
- Segments - large free extents for writing new data. LFS extends journaling to data (hardware RAID can do this too though)
- Checkpoints, roll-forwards are part of this.
- Traditional FSs have integrity problems but that is solved by Journaling. Journaling is:
- + Fast
- + Stable