Design Document for the Buddy File System

By Bryan Pass, Emily Fetchko, and Rick Carback

Introduction

BuddyFS is a distributed file system designed to provide the benefits from a centralized design without a single point of failure or bottleneck. It combines two topologies, a two-level hierarchical structure for file indexing and a clustering structure for file storage groups. These file storage groups will handle the digital rights management. BuddyFS's main goals are high availability of files and low latency times for file access and manipulation. File consistency will also be protected.

Assumptions

1. All of the top level nodes won't die at once. At any time, a subset of the nodes will be designated as top level nodes and will handle file indexing for the system. If any subset of these would die, then the others would still have the list of files and new top level nodes would be chosen. However, if all were to die at once, the complete list of files would be lost.

2. The system contains less than 1000 machines. Since all machines need to keep TCP connections established with other machines, we feel 1000 nodes is a reasonable upper limit for the system.

3. Multicast isn't available. When an update is sent to multiple nodes, a copy is sent for each node. Multicast would make this process more efficient, but most routers block multicast packets.

4. All nodes are trusted. No node in the system will deliberately sabotage files or not follow the established protocols.

5. Nodes need to be connected to participate. Our file system does not have a disconnected mode as Coda does.

Design Objectives

1. High availability of files. Even when nodes fail files should still be available whenever possible. This goal will be achieved through the replication of files and file indexes.

2. Low latency times for finding and manipulating files. Users should be able to access and store files immediately just as if the files were on their own hard drives. This goal will be achieved by minimizing the number of messages sent.

3. Scalability. As the system grows performance should degrade gracefully. This system should work well for tens or even hundreds of nodes. The two level and clustering structures will help to achieve this goal since all file management is distributed across the system.

4. Preservation of Unix file semantics. Users should be able to read and write files just as if they were on their own hard drive. When a user downloads a file, it should be the most recent copy available, and when a user writes, the write should be committed to all copies of the file as soon as possible. File storage groups communicate within each other to keep files updated and writes serialized.

Design

Nodes in the file system are organized in a two level structure, with a logical "top" level and "bottom" level. Top level nodes are responsible for communicating directly with each other to handle nodes joining and leaving the system and file creation and deletion. Top level nodes also keep an index so that other nodes can locate files in the system. Bottom level nodes keep a list of all of the top level nodes in the system.

When a node first joins the system, it establishes a TCP connection to any previously connected node. This previously connected node then provides the list of top level node addresses to the new node. If the node that sent the list was not in the list itself, the new node will disconnect from that node and connect randomly to a top level node. Once connected, the node will send the list of files it has to the top level node. The top level node will add these files to the index it keeps.

Every node in the system keeps a list of all top level nodes. When a top level node is added or deleted, all nodes in the system are given this information by the top level node they are connected to. Thus, if a node loses its top level node, it will reconnect to some other top level node in its list. As stated in the assumptions section of this document, the system cannot handle an instance when all top level nodes leave the system at the same time, or nearly the same time. Should this event occur, the system will enter an undefined state and may require human intervention to restore it to a working state.

Depending on the Digital Rights associated with each file, it is possible for a file to have many copies (called “replicas”) on different nodes in the system. Each node containing a replica of a given file establishes connections with all other nodes containing replicas of the same file, forming a completely connected graph. When a node wishes to access a file that has been replicated, it connects to any node in the file group and requests access. The nodes that own the file communicate with each other to manage locks and consistency for the file. Whenever a file replica is created or deleted, or when a node disconnects from the system, a top level node is notified of the change. This top level

node then forwards the information to the other top level nodes so that they can all maintain an up-to-date list of nodes and files in the system.

A node that stores replicas may temporarily disconnect and reconnect to the system. During the initial connection phase to a top level node, the top level node will recognize which files are replicas of files other nodes own. The top level node will then respond with node addresses for file groups which the reconnecting node should join. This node then connects to each of those groups and synchronizes its replica of the file with the other replicas already present in the system. This synchronization may result in removal from the group because of inconsistencies, or a forced update to a more recent version of the file, either for the joining node or the other members in the group.

Digital Rights Management

As with regular UNIX file systems, each file has an owner and a group. The owner of the file may modify the digital rights options for the file. Any of the following digital rights may be associated with the file and changed by the owner. The owner may also choose to allow anyone in the file's owning group to make changes to the digital rights.

Number of Replicas
Allowed/Disallowed Sites where replicas may be stored
Readable (Owner/Group/Anyone)
Writable (Owner/Group/Anyone)
Appendable (Owner/Group/Anyone)
Removable (Owner/Group/Anyone)

Digital rights enforcement is the responsibility of the node where the file is stored. When a request is received, the node verifies that the username, group name, and password supplied are correct and then the digital rights for the file are checked to ensure the requested operation may be performed. This approach is totally decentralized, but as a result it requires that every node in the system be trusted to respect the rights associated with all the files for which it stores a replica. To prevent users from bypassing the digital rights management by accessing a file that may be located on their disk, protected files will be stored using an encryption mechanism.

Implementation

The system will be implemented as a Linux kernel virtual file system driver using the "File System in User Space" project (fuse.sourceforge.net). This implementation will allow more rigorous testing because the file system can be used by any Linux application, rather than those specifically designed for testing this project. A set of command line tools will be created to allow modification to the digital rights associated with each file.

Testing Strategy

Our test strategy involves creating a small scale model of the system to store and modify files. We will use the graduate student lab and create at least 10 clients for the system. We will also create an assortment of files of different sizes ranging from 100 bytes to 10 MB. These files will be created by a script. Each of the ten clients will then download and modify random files. Clients will demonstrate open, read, write, copy, close, delete, and any other supported functionality. Files will use different settings for digital rights management and replication. We will monitor these file transactions and check for consistency errors or DRM violations.

Proposed Performance Evaluation

During testing, we will monitor the performance of the system. The test scripts will allow for different sizes of files to be created and downloaded at different rates. We will increase the speed of file accesses and monitor the the load on the nodes and the network. We will also run similar tests on a single user Unix file system and on the UMBC AFS system and compare the results.

Timeline

Phase 1:
3/7/05: Preliminary meeting
3/16/05: Design meeting
3/21/05: Design meeting #2
3/25/05: VFS written
3/30/05: Final design meeting
4/1/05: Design document completed

Phase 2:
4/6/05: Project status meeting
4/13/05: Midterm status meeting
4/20/05: Midterm status report completed
4/22/05: Basic file system completed (no replication or DRM)

Phase 3:
4/27/05: Project status meeting
4/29/05: DRM completed
5/6/05: Replication completed
5/11/05: Project status and test meeting
5/13/05: Testing completed
5/17/05: Final report done