Disconnected Operation: Extending YFS to handle network disconnectivity and conflict resolution

Owen Derby and Rob McQueen

May 14, 2012

1 Problem Statement

Yet-Another-Filesystem (YFS) is a distributed filesystem that allows clients across a network to modify the same filesystem. YFS enables clients to cache content locally (we branched from lab 5). However, YFS is restrictive; it does not allow clients to modify the filesystem while disconnected from the extent and lock servers. To solve this problem, we have extended YFS with a file versioning system, client-side conflict detection and resolution, and stale locking. This allows clients to modify cached content while disconnected from the servers.

2 Design

2.1 File Versioning System

To enable conflict detection, we maintain a totally ordered version number for every file in the filesystem. The extent server increments the version number any time a client puts an extent to the server. Since clients only flush extents to the extent server if they've been modified, it follows that the version number of a given extent is incremented once for every modification made to that file. The extent server ensures that no old/stale data is written by rejecting any extents put without the current version number.

2.2 Stale Locking

To handle the case when clients disconnect while holding locks, we introduce the concept of stale locks. The owner of a lock is defined by the lock server. This causes a problem in the case of disconnected operation, as the client can no longer know if it owns a lock or not. When the client doesn't know whether it owns a lock or not, we call it a "stale lock".

To ensure eventual consistency, a disconnected client marks all of its locks as stale upon reconnecting to YFS. Upon (re)acquiring a lock from the lock server that was perviously stale, a client needs to resolve conflicts for the corresponding extent. This is accomplished using the file versioning system, as described below.

2.3 Conflict Detection and Resolution

Conflict detection and resolution is performed entirely on the client-side. A conflict in a file occurs if two clients modified the same version of the same file (because one or both were disconnected and using stale locks).

As mentioned above, when a client reconnects to YFS it reacquires all stale locks and proceeds to search for conflicts. A client knows absolutely if a conflict has occurred, once it has the lock, if the version number on the extent server is not the same as the version number it has cached locally and the local cached copy is dirty. If it's local copy is not dirty, then there is no conflict and it will simply replace its copy with the server's copy, regardless of version. If the versions are the same, but its cached copy is dirty, it does nothing (keeping it's dirty version in the cache) and assumes it will flush its changes to the server when the lock is revoked.

When a conflict is detected, the client will save its current version of the file as a new file (thus allocating a new inode and permanently placing it in YFS) to act as a temporary copy of the conflicting file. Then it will recover the most recent version from the extent server and system operation to continue. Resolution of the conflict is then done manually by the user application. This ensures that the client quickly resolves all stale locks and that it is updated on the latest state of YFS. Continual disconnection and reconnection with conflicting modifications will thus only create new temporary files, one for each conflict, but never cause for incorrect state of the extent.

2.4 Example Case

Assume a system of two YFS Clients, C1 and C2, a lock server, LS, and an extent server, ES. Both clients are connected; C1 holds lock L1 corresponding to extent E1.

Client C1 disonnects. The LS detects that C1 is disconnected and grants L1 to C2. C2 then modifies E1. Meanwhile, C1 also modifies E1, creating conflicting versions of the file.

C1 reconnects, marks L1 as stale, and checks for conflicts on E1 by reacquiring L1 from. LS revokes L1 from C2 and tells C1 to retry. C1 retries and acquires L1 from LS and then asks ES for its version of E1. Seeing that the remote version is different, and that it has local modifications to E1, C1 must resolve the conflict, as described above.

3 Conclusion

YFS is restrictive in that it does not allow clients to modify the filesystem while disconnected. By adding a file versioning system, client-side conflict resolution, and stale locking, we have extended YFS so that clients can modify extents while in a disconnected state. File versioning allows for the detection of conflicts between extents. Clients handle conflict resolution by creating a temporary file with the conflicting copy of the extent, and giving the user application the respondibiliity for merging conflicts. Stale locking enabled clients to modify extents while disconnected and resolve the extents upon reconnection.