Distributed File Systems, Continued

CS 239, Spring 1996

May 22, 1996

Outline

- Basic concepts
- NFS
- Andrew File System
- Replicated file systems
  - Ficus
  - Coda
- Serverless file systems

Replicated File Systems

NFS provides remote access

AFS provides high quality caching

Why isn't this enough?

More precisely, when isn't this enough?

When Do You Need Replication?

- For write performance
- For reliability
- For availability
- For mobile computing
- For load sharing

Optimistic replication increases these advantages

Some Replicated File Systems

- Locus
- Ficus
- Coda
- Rumor

All optimistic - few conservative file replication systems have been built

Ficus

Optimistic file replication based on *peer-to-peer* model

Built in Unix context

Meant to service large network of workstations

Built using stackable layers

Peer-To-Peer Replication

All replicas are equal

No replicas are masters, or servers

All replicas can provide any service

All replicas can propagate updates to all other replicas

Client/server is the other popular model

Basic Ficus Architecture

Ficus replicates at volume granularity

Given volume can be replicated many times

- Performance limitations on scale

Updates propagated as they occur

- On single best-efforts basis

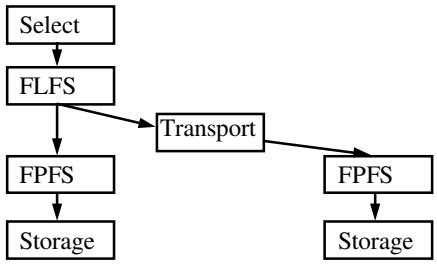
Consistency achieved by periodic reconciliation

Stackable Layers in Ficus

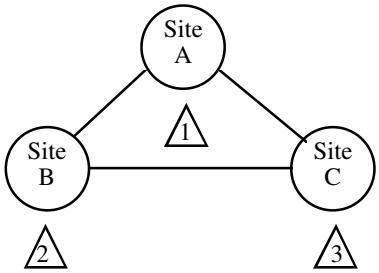
Ficus is built out of several stackable layers

Exact composition depends on what generation of system you look at

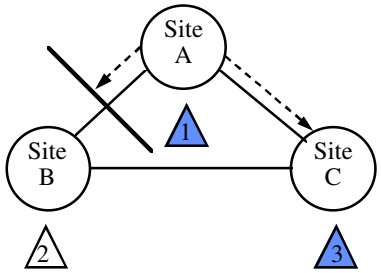
Ficus Stackable Layers Diagram



Ficus Diagram



An Update Occurs



Reconciliation in Ficus

Reconciliation process runs periodically on each Ficus site

For each local volume replica

Reconciliation strategy implies eventual consistency guarantee

Frequency of reconciliation affects how long "eventually" takes

Steps in Reconciliation

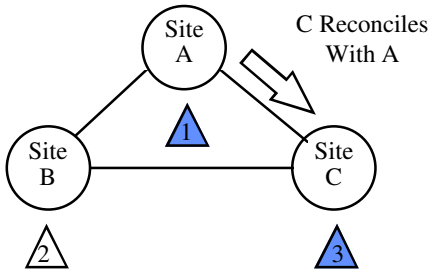
1). Get information about the state of a remote replica

2). Get information about the state of the local replica

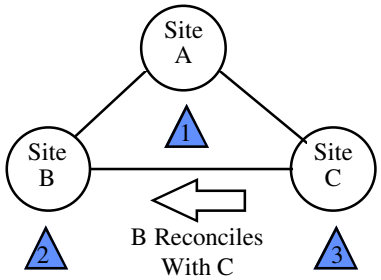
3). Compare the two sets of information

4). Change local replica to reflect remote changes

Ficus Reconciliation Diagram



Ficus Reconciliation Diagram, Con't



Gossiping and Reconciliation

Reconciliation benefits from the use of gossip

In example just shown, an update originating at A got to B through communications between B and C

So B can get the update without talking to A directly

Benefits of Gossiping

- Potentially less communications
- Shares load of sending updates
- Easier recovery behavior
- Handles disconnections nicely
- Handles mobile computing nicely

Peer model systems get more benefit than client/server model systems

Reconciliation Topology

Reconciliation in Ficus is pairwise

In the general case, which pairs of replicas should reconcile?

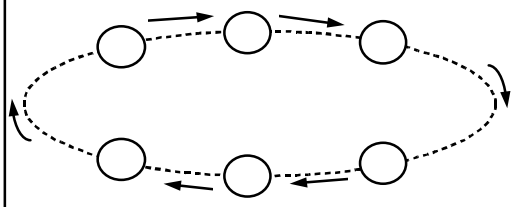
Reconciling all pairs is unnecessary

Due to gossip

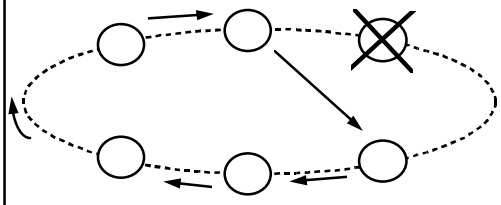
Want to minimize number of recons

But propagate data quickly

Ficus Ring Reconciliation Topology



Adaptive Ring Reconciliation Topology



Problems in File Reconciliation

- Recognizing updates
- Recognizing update conflicts
- Handling conflicts
- Recognizing name conflicts
- Update/remove conflicts
- Garbage collection

Ficus has solutions for all these problems

Recognizing Updates in Ficus

Ficus keeps per-file version vectors

Updates detected by version vector comparisons

The data for the later version can then be propagated

Ficus propagates full files

Recognizing Update Conflicts in Ficus

Concurrent update can lead to update conflicts

Version vectors permit detection of update conflicts

Works for n-way conflicts, too

Handling Update Conflicts in Ficus

Ficus uses *resolver* programs to handle conflicts

Resolvers work on one pair of replicas of one file

System attempts to deduce file type and call proper resolver

If all resolvers fail, notify user

Ficus also blocks access to file

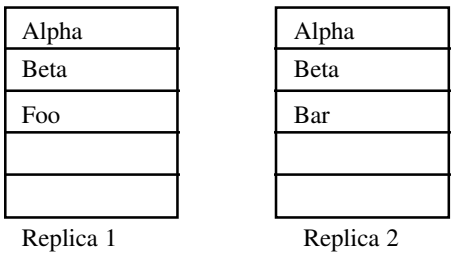
Handling Directory Conflicts in Ficus

Directory updates have very limited semantics

So directory conflicts are easier to deal with

Ficus uses special in-kernel mechanisms to automatically fix most directory conflicts

Directory Conflict Diagram



How Did This Directory Get Into This State?

If we could figure out what operations were performed on each side that caused each replica to enter this state,

We could produce a merged version

But there are two possibilities

Possibility 1

1). Alpha and Beta exist

2). Create Foo at replica 1

3). Create Bar at replica 2

Correct result is directory containing Alpha, Beta, Foo, and Bar

The Create/Delete Ambiguity

This is an example of a general problem with replicated data

Cannot be solved with per-file version vectors

Requires per-entry information

Ficus keeps such information

Must save removed files' entries for a while

Possibility 2

1). Alpha, Beta, and Foo exist

2). Delete Foo at replica 2

3). Create Bar at replica 2

Correct result is directory containing Alpha, Beta, and Bar

And there are other possibilities

Recognizing Name Conflicts in Ficus

Name conflicts occur when two different files are concurrently given same name

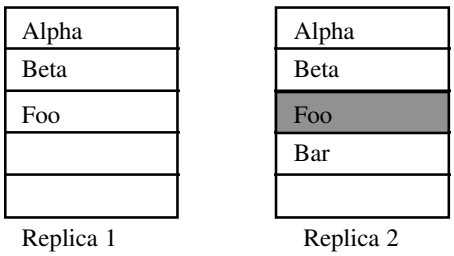
Ficus recognizes them with its per-entry directory info

Then what?

Handle similarly to update conflicts

Add disambiguating suffixes to names

Internal Representation of Problem Directory



Update/Remove Conflicts

Consider case where file "foo" has two replicas -

1). Replica 1 receives an update

2). Replica 2 is removed

What should happen?

A matter of systems semantics, basically

Ficus' No-Lost-Updates Semantics

Ficus handles this problem by defining its semantics to be no-lost-updates

In other words, the update must not disappear

But the remove must happen

Put "foo" in the orphanage

Requires temporarily saving removed files

Removals and Hard Links

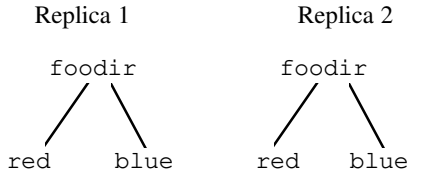
Unix and Ficus support hard links

Effectively, multiple names for a file

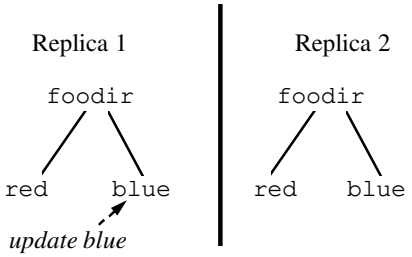
Cannot remove a file's bits until the last hard link to the file is removed

Tricky in a distributed system

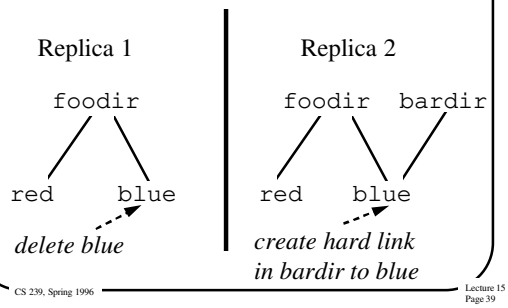
Link Example



Link Example, Part II



Link Example, Part III



What Should Happen Here?

- Clearly, the link named `foodir/blue` should disappear
- And the link in `bardir` should remain
- But what version of the data should the `bardir` link point to?

No-lost-update semantics say it must be the update at replica 1

Garbage Collection In Ficus

Ficus cannot throw away removed things at once

- Directory entries
- Updated files for no-lost-updates
- Non-updated files due to hard links

When can Ficus reclaim the space these use?

When Can I Throw Away My Data?

Not until all links to the file disappear

Global information, not local

Moreover, just because I know all links have disappeared doesn't mean I can throw everything away

Must wait till everyone knows

Requires two trips around the ring

Why Can't I Forget When I Know There Are No Links?

I can throw the data away

I don't need it, nobody else does either

But I can't forget that I knew this

Because not everyone knows it

For them to throw their data away, they must learn

So I must remember for their benefit

Coda

A different approach to optimistic replication

Inherits a lot from Andrew

Basically, a client/server solution

Developed at CMU

Coda Replication Model

Files stored permanently at server machines

Client workstations download temporary replicas, not cached copies

Can perform updates without getting token from the server

So concurrent updates possible

Detecting Concurrent Updates

Workstation replicas only reconcile with their server

At recon time, they compare their state of files with server's state

Detecting any problems

Since workstations don't gossip, detection is easier than in Ficus

Handling Concurrent Updates

Basic strategy is similar to Ficus'

Resolver programs are called to deal with conflicts

Coda allows resolvers to deal with multiple related conflicts at once

Also has some other refinements to conflict resolution

Server Replication in Coda

Unlike Andrew, writeable copies of a file can be stored at multiple servers

Servers have peer-to-peer replication

Servers have strong connectivity, crash infrequently

Thus, Coda uses simpler peer-to-peer algorithms than Ficus must

Why Is Coda Better Than AFS?

- Writes don't lock the file
  - Writes happen quicker
  - More local autonomy
- Less write traffic on the network
- Workstations can be disconnected
- Better load sharing among servers

Comparing Coda to Ficus

- Coda uses simpler algorithms
  - Less likely to be bugs
  - Less likely to be performance problems
- Coda doesn't allow client gossiping
- Coda has built-in security
- Coda garbage collection simpler

Serverless Network File Systems

New network technologies are much faster, with much higher bandwidth

In some cases, going over the net is quicker than going to local disk

How can we improve file systems by taking advantage of this change?

Fundamental Ideas of Serverless File Systems

- Peer workstations providing file service for each other
- High degree of location independence
- Make use of all machine's caches
- Provide reliability in case of failures

xFS

Serverless file system project at Berkeley

Inherits ideas from several sources
  - LFS
  - Zebra (RAID-like ideas)
  - Multiprocessor cache consistency

Built for Network of Workstations (NOW) environment

What Does a File Server Do?

- Stores file data blocks on its disks
- Maintains file location information
- Maintains cache of data blocks
- Manages cache consistency for its clients

xFS Must Provide These Services

In essence, every machine takes on some of the server's responsibilities

Any data or metadata might be located at any machine

Key challenge is providing same services centralized server provided in a distributed system

Key xFS Concepts

- Metadata managers
- Stripe groups for data storage
- Cooperative caching
- Distributed cleaning processes

How Do I Locate a File in xFS?

I've got a file name, but where is it?

Assuming its not locally cached

File's directory converts name to a unique index number

Consult the metadata manager to find out where file with that index number is stored - the *manager map*

The Manager Map

Data structure that allows translation of index numbers to file managers

Not necessarily file locations

Kept by each metadata manager

Globally replicated data structure

Simply says what machine manages the file

Using the Manager Map

Look up index number in local map

Index numbers are clustered, so many fewer entries than files

Send request to responsible manager

What Does the Manager Do?

Manager keeps two types of information

1). imap information

2). caching information

If some other site has the file in its cache, tell requester to go to that site

Always use cache before disk

Even if cache is remote

What If No One Caches the Block?

- Metadata manager for this file then must consult its *imap*
- Imap tells which disks store the data block
- Files are striped across disks stored on multiple machines
  - Typically single block is on one disk

Writing Data

- xFS uses RAID-like methods to store data
- RAID sucks for small writes
- So xFS avoids small writes
- By using LFS-style operations -
  - Batch writes until you have a full stripe's worth

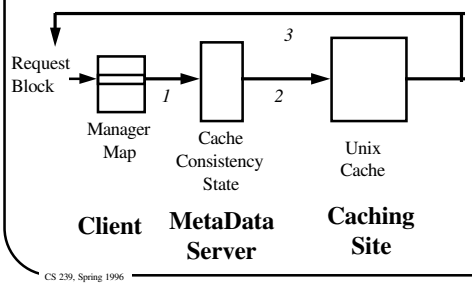
Stripe Groups

- Set of disks that cooperatively store data in RAID fashion
- xFS uses single parity disk
- Alternative to striping all data across all disks

Cooperative Caching

- Each site's cache can service requests from all other sites
- Working from assumption that network access is quicker than disk access
- Metadata managers used to keep track of where data is cached
  - So remote cache access takes 3 network hops

Getting a Block From a Remote Cache



Providing Cache Consistency

- Per-block token consistency
- To write a block, client requests token from metadata server
- Metadata server retrieves token from whoever has it
  - And invalidates other caches
- Writing site keeps token

Which Sites Should Manage Which Files?

- Could randomly assign equal number of file index groups to each site
- Better if the site using a file also manages it
  - In particular, if most frequent writer manages it
- Can reduce network traffic by ~50%

Cleaning Up

- File data (and metadata) is stored in log structures spread across machines
- A distributed cleaning method is required
- Each machine stores info on its usage of stripe groups
- Each cleans up its own mess

Basic Performance Results

- Early results from incomplete system
- Can provide up to 10 times the bandwidth of file data as single NFS server
- Even better on creating small files
- Doesn't compare xFS to multimachine servers