Peer-to-Peer File Systems

Hannes Geissbühler
Seminar of Distributed Computing
WS 03/04

The Papers

- Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications
  Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Blakrishnan
  MIT Laboratory for Computer Science
- Wide-area cooperative storage with CFS
  Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica
  MIT Laboratory for Computer Science
- Ivy: A Read/Write Peer-to-Peer File System
  Athicha Muthitacharoen, Robert Morris, Thomer M. Gil and Benjie Chen
  MIT Laboratory for Computer Science

Goal of this Talk

- To show how to build a peer-to-peer file system based on these three papers
- To explain how the different layers of this peer-to-peer system work
- Point out the problems of peer-to-peer file systems

Chord, a Distributed Lookup Protocol - Overview

- Provides support for just one operation: given a key it maps the key onto a node (IP-address)
- Chord is a scalable protocol for lookup in a dynamic peer-to-peer system with frequent node arrivals and departures
- Chord uses a variant of consistent hashing to assign keys to Chord nodes

Chord, Runtime Aspects

- Needs routing information about only O(log N) other nodes
- Resolves lookups via O(log N) messages to other nodes
- Maintaining routing information as nodes join and leave results in no more than $O(\log^2 N)$ messages
- When an N^{th} node joins or leaves the network only an O(1/N) fraction of keys are moved to a different location

Advantage of Chord compared to other Systems

- Simple
- Provable correct
- Provable performance
- Robust (in face of partially incorrect routing information)
- Handles concurrent node joins and failures

System Model

- Load balance: Chord acts as a distributed hash function ->keys are evenly spread over the nodes
- Decentralization: Chord is fully distributed. No node is more important than any other ->robustness
- Scalability: The cost of lookup grows as the log of the number of nodes ->large systems are feasible
- Availability: Chord automatically adjusts its internal tables -> nodes can always be found even when major failures in the network occur

What Chord is not responsible for

- Authentication
- Caching
- Replication
- User-friendly naming of data

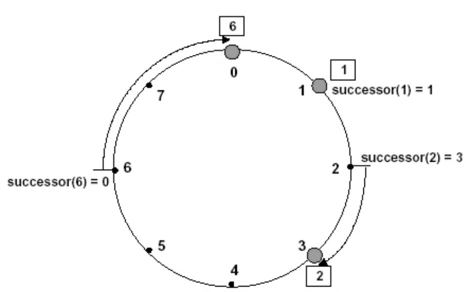
Consistent Hashing

- Assigns each node and key an m-bit identifier
- A node's identifier is chosen by hashing the node's IP address
- A key identifier is produced by hashing the key

Assigning Keys to Nodes 1

- Identifiers are ordered in an identifier circle modulo 2^m
- Key k is assigned to the first node whose identifier is equal to or follows k on the circle. This node is called the successor node
- In a circle of numbers from 0 to 2^m-1, successor(k) is the first node clockwise from k

Example Identifier Circle



Assigning Keys to Nodes 2

- When a node n joins, certain keys previously assigned to n's successor now become assigned to n
- When node n leaves the network, all of its assigned keys are reassigned to n's successor
- Having N nodes and K keys following theorems hold:

  -Each node is responsible for at most (1+ε)K / N keys

  -When an $(N+1)^{th}$ node joins or leaves the network only responsibility for O(K / N) keys changes

  One can prove a bound of ε = O(log N)

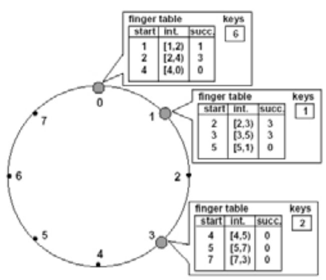
Key Location (Routing)

- Each node need only be aware of its successor
  ->queries can be passed around the circle
  ->inefficient
- Chord maintains additional routing information
- Additional information is not essential for correctness, which is achieved through the correct successors

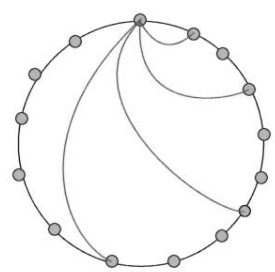
Additional Routing Information - The Finger Table

- Each node maintains a routing table with at most m entries called the finger table
- The i^{th} entry in the table at node n contains the identity of the first node s that succeeds n by at least 2^{i-1} on the circle
- S = successor(n+ 2^{i-1})
- We call node S the i^{th} finger
- A finger table entry includes the Chord identifier and the IP address

Example Finger Table



Finger Table and Search



Searching a Node

- Searching recursively the successor of a key over the finger nodes.
- Every lookup roughly halves the distance on the circle
- The number of contacted nodes to find a successor in a N-node network is O(log N)

Node Joins

- 2 invariants have to be preserved
  - Each node's successor is correct
  - node successor(k) is responsible for k
- To simplify joins and leaves a predecessor pointer is maintained

A Node Join

- 1. initialize the predecessor and fingers of node n
- 2. update the fingers and predecessors of existing nodes
- 3. move responsibility of keys

Stabilization

- Having failures and concurrent operations the join algorithm discussed is to aggressive
- Use stabilization protocol to keep nodes' successors pointers
- Every node runs stabilize periodically

Failures

- Key is to maintain correct successor pointers
- Each Chord node maintains a successor-list of its r nearest successors
- Good length of successor list is log N

Problems of Chord

- Partitioned rings
- Malicious participants
- Lookup latency

Chord Summary

- Just one operation: maps key onto node
- Simple, correct, scalable
- In simulations all good properties have been verified

CFS, the Cooperative File System - Overview

- Peer-to-peer read only storage system
- Provides distributed hash table for block storage (DHash)
- Uses replicas and caches blocks

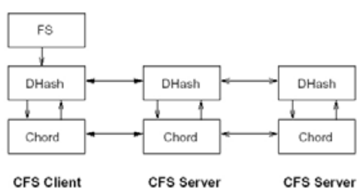
CFS File System

- File system exists as a set of blocks distributed over available nodes
- CFS client interprets blocks as file system

DHash

- Splits files into blocks and distributes them
- Maintains cached blocks and replicated copies
- Supports pre-fetching of cached blocks to decrease latency

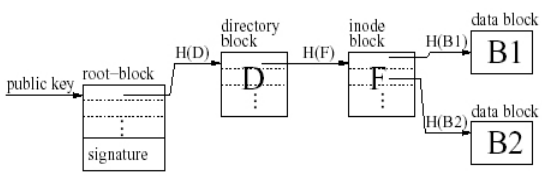
Software Structure



File system format

- Each block is a piece of file or a piece of meta-data (for example a directory)
- Size of a block is in order of tens of kB
- Publisher inserts root block signed with private key
- Data is stored during a finite interval
- No explicit delete operation

File System Structure Example



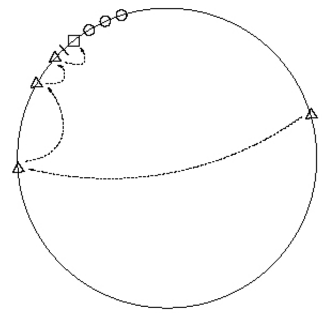
Replication

- DHash replicates each block on k other servers to increase availability
- Replicas are maintained as peers come and go
- Replicas are placed on the successor servers (easy to find through successor list)

Caching

- DHash cashes blocks
- Each server has a fixed amount of disk storage reserved for cache
- Least recently used replacement
- Each node on the lookup path gets a cache copy
- While searching each server has to check if the desired block is cached on the node

Caching Example



Load Balance

- DHash spreads the blocks evenly around the ID space (hash function)
- To accommodate different server capacities virtual servers are used
- Virtual servers have direct access to other servers on the same machine
- Number of servers can dynamically be adapted

Problems of CFS

- DHash -> saving time is limited
- No explicit delete operation
- Virtual servers -> nodes are not independent
- One small change in a file causes big effort in rearranging the data structure
- Problems of Chord

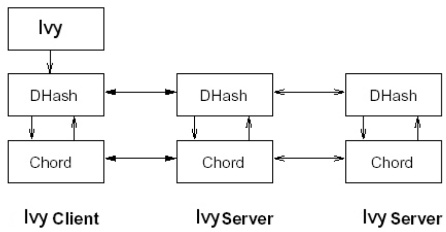
Cooperative File System Summary

- Highly scalable read only file system
- Clients retrieve blocks from servers and interpret them as a file system
- CFS uses caching and replication
- Experimental results show that CFS is as fast as FTP

Ivy, a Read / Write Peer-to-Peer File System - Overview

- Ivy provides NFS-like semantics
- Ivy consists solely of a set of logs
- Resists attacks from non-participants by cryptographically verifying the data

Layers of Ivy



Design

- Ivy consists of a set of logs, one log per participant
- Each participant appends only its own log but reads from all logs
- Each participant maintains a private snapshot to avoid going through all logs

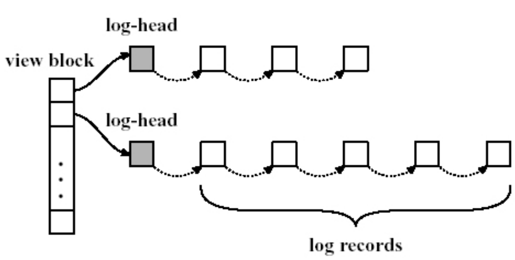
Log Data Structure

- A log is a linked list of immutable log records
- Each log is a DHash block
- Log-head stores DHash key of most recent log record
- Log records contain the Inumber(s) of the file(s) or directory they affect
- Log records contain a version vector to guarantee causality

Views

- Participants agree on a view
- Users creating or changing a file system must exchange public keys
- View block = pointers to all log-heads & root Inode

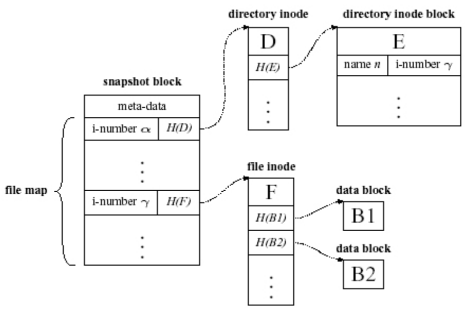
View and Logs



Using the Log

- File system creation
  -create log(s), log–head(s), root Inode and the view
- File creation
  -append Inode log record
- File read
  -scans all logs for records concerning the Inode

Snapshot



Application Semantics 1

- Ivy defers writing file data until the application is closing the file
  ->only once per file-write a new log-head is inserted
- updates can occur in order or at one time
  ->difficult for a decentralized file system
  if version vectors are equal -> comparing participants public keys

Application Semantics 2

- Combination of deletion and renaming
  P1 wants to delete file a
  P2 wants to rename file a to b
  Ivy will return a success status to both, but the system agrees on the version vector order

Problems of Ivy

- One has to save the logs forever
- Unsolvable conflicts
- Problems of underlying layers

Summary

- Ivy is a peer-to-peer file system which is built on top of Chord and DHash
- Ivy can operate a relatively open peer-to-peer environment
- Experimental results show that Ivy is two to three times slower than NFS