

P2P: Advanced Topics Filesystems over DHTs and P2P research

Vyas Sekar

DHT Overview

- DHTs provide a simple primitive
 - ♦ put (key,value)
 - ♦ get (key)
 - ♦ Data/Nodes distributed over a key-space
- High-level idea: Move closer towards the object in the key-space
 - ♦ Typically $O(\log N)$ lookup time
 - ♦ Typically $O(\log N)$ neighbors in routing table

How to build applications over DHTS

- DHTs provide mechanisms for retrieval
 - ♦ Chord etc. don't store keys and values
- How to build a file-system over DHTs?
 - ♦ Practical implementation issues
 - Storage granularity
 - Replication
 - Network latency
 - Reliability
 -
- Case Study: CFS

What we would like to have ..

- Decentralized control
 - ♦ Ordinary internet hosts
- Scalability
 - ♦ Performance should scale with #nodes
- Availability
 - ♦ Resilience to node failures
- Load balancing
 - ♦ Irrespective of workload/node capacities

CFS Design

Layer	Responsibility
FS	Interprets blocks as files; presents a file system interface to applications.
DHash	Stores unstructured data blocks reliably.
Chord	Maintains routing tables used to find blocks.

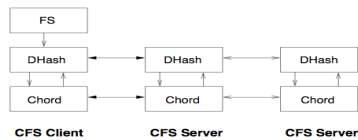


Figure 1: CFS software structure. Vertical links are local APIs; horizontal links are RPC APIs.

Design Overview

- Filesystem structure
 - ♦ Similar to UNIX
 - ♦ Instead of disk blocks/addresses use DHash block and block identifies
 - Each block is either data or meta-data
 - ♦ Parent block contains ids of children
- Insert the file blocks into CFS
 - ♦ Hash of each blocks content as its id
 - ♦ Root blocks need to be signed for integrity

CFS file system structure

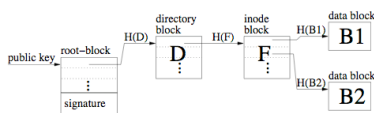


Figure 2: A simple CFS file system structure example. The root-block is identified by a public key and signed by the corresponding private key. The other blocks are identified by cryptographic hashes of their contents.

Chord: Server selection

- Original Chord idea only guarantees $O(\log N)$ lookup time
 - ♦ But these $O(\log N)$ could be very long physical latencies
 - E.g., Inter-continental links!
- Server selection added in CFS
 - ♦ Estimate total cost of using each node in the set of potential next hops
 - ♦ Assume latencies are transitive

$$C(n_i) = d_i + \bar{d} \times H(n_i)$$

$$H(n_i) = \text{ones}((n_i - id) >> (160 - \log N))$$

DHash Layer

- Performs fetches for clients
- Distributes file blocks among servers
- Maintains cached and replicated copies
- Load balance: Control over how much data each server stores
- Management: Quotas/Updates/Deletes

DHash Design: Block vs. File ?

- What granularity to store file-system objects?
- File-level storage
 - + lower lookup cost
 - load imbalance
- Block-level storage
 - + balance load, efficient for large popular files
 - more lookups per file
- CFS chooses block-level storage
 - ♦ Network bandwidth for lookup is low
 - ♦ Lookup latency is hidden by pre-fetching
 - ♦ File-level storage uses capacity inefficiently

DHash: Replication

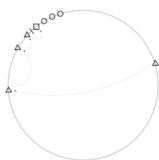


Figure 4: The placement of an example block's replicas and cached copies around the Chord identifier ring. The block's ID is shown with a tick mark. The block is stored at the successor of its ID, the server denoted with the square. The block is replicated at the successor's immediate successors (the circles). The hops of a typical lookup path for the block are shown with arrows; the block will be cached at the servers along the lookup path (the triangles).

- Replicate each block on k servers
 - ♦ After the successor on the ring
- Independence
 - ♦ Servers close on key-space unlikely to be correlated in network
- Also makes it easy to speed up downloads

DHash Caching

- Caching to avoid hot-spots
- When a lookup succeeds, the client sends a copy to each server along the Chord-path
 - ♦ Just second-to-last server?
 - ♦ At each lookup step, check if cached copy exists
- Replacement: Least recently used policy
 - ♦ Blocks farther away in ID-space likely to be discarded earlier
- Cache consistency not a problem!
 - ♦ Data is indexed by a content-hash

DHash: Load Balance

- Servers may have different network and storage capacities
- Introduce notion of “virtual server”
 - ♦ Each physical node may host multiple CFS servers
 - ♦ Configured depending on capacity
- May increase lookup latency?
 - ♦ Shortcuts by sharing Chord routing information

DHash: Management

- Quotas to prevent abuse
 - ♦ Per-IP limit data you can enter into CFS
- Updates
 - ♦ Read-only semantics
 - ♦ Only publisher can update the file system
- Deletes
 - ♦ No explicit delete: data stored for an agreed interval and discarded unless explicitly requested
 - ♦ Automatically gets rid of malicious inserts

Some interesting P2P/DHT topics

- Different geometries possible
 - ♦ Ring e.g., Chord
 - ♦ Hypercube e.g., CAN
 - ♦ Prefix-trees e.g., PRR trees
 - ♦ Butterfly network e.g., Viceroy

Some interesting P2P/DHT topics

- Can we do better than the $O(\log N)$ time?
 - ♦ Use aggressive caching
 - ♦ Use object popularity
- Can we do better than $O(\log N)$ space?

Some interesting P2P/DHT topics

- Can we add locality to DHTs?
 - ♦ Support range-queries
 - ♦ So that file blocks get hashed to nearby locations
 - Locality-sensitive hashing
 - ♦ Network latency to be explicitly modeled

Some interesting P2P/DHT topics

- P2P streaming?
 - ♦ ESM etc. take the approach of building "latency-optimized" overlays instead of general purpose DHTs
 - ♦ Why -- DHTs don't have network locality

Some interesting P2P/DHT topics

- Anonymous storage
 - ♦ Freenet
 - ♦ Publius
 - ♦ Probabilistic routing and caching
 - ♦ Cant predict where the object came from!