Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica
Wide-area cooperative storage with CFS
In the Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01), Chateau Lake Louise, Banff, Canada. October 2001

1. What were the goals of CFS? What applications did they target?

→ Take failures + churn to extreme

- decentralized control: servers vs. publishers
- scalability — O(log N) lookups
- availability — Robust to crashes + CHURN
- load balance
- persistence
- quotas
- efficiency (log N lookup)

---

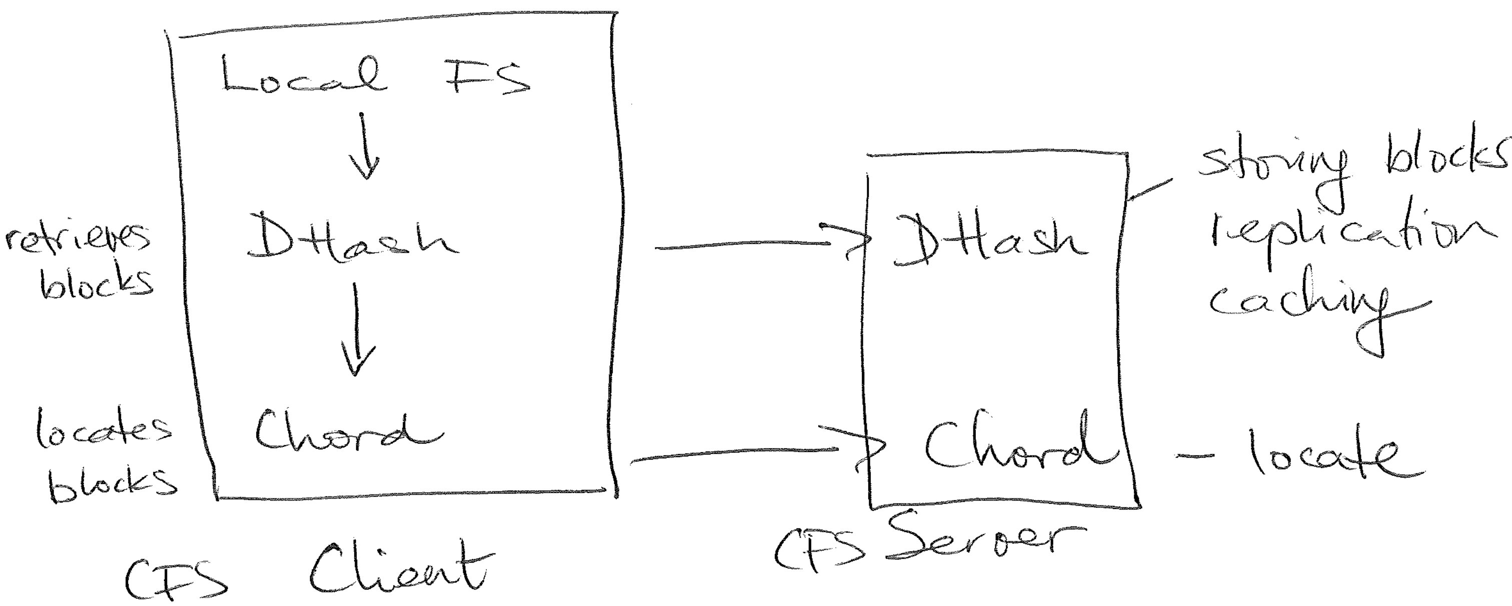
not well described

- write once, read many } Applications
-

publisher

clients

2. What is the overall architecture of their system? (i.e., what are the three layers and what are their responsibilities?) Is there a clean separation between layers?



No clean layer

DHash uses chord algorithm,

but needs to cache along way

3. Figure 2: How is a CFS file system structured? How does a publisher insert blocks into CFS? What happens if a publisher wants to update blocks? How and why is the root block treated differently?

Fig. 2

- up from leaves to root

- lookup by content hash - Merkle tree

- Root: new block <u>signed</u> by same private key
- lookup w/ public key
- timestamp

4. Figuring out the correct way to deal with delete operations can be tricky in distributed systems. Why is having an explicit delete non-trivial in distributed systems? How have other systems (e.g., Google FS) dealt with the possible problem? In contrast, CFS chose to, in effect, ``lease'' storage space; the publisher can repeatedly ask for extensions. What are the advantages of this approach? Disadvantages?

- Tricky if miss deletes - not reacable or down
- GFS used soft state/ garbage collection

- Leases:

  Advt Both sides know situation even if not readable

  + Can't miss operations
  + Can take away eventually...

  Dis
  - If can't reach, lose data!

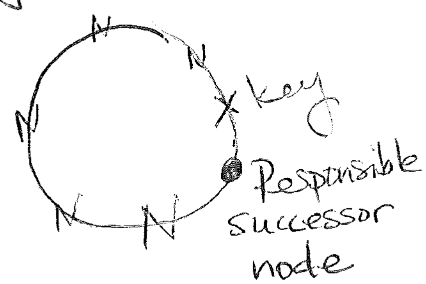
5. CFS uses Chord to locate blocks on a specific node based on the content-hash of the data to a key. The Chord lookup layer, overlay network, was introduced in SIGCOMM'01 paper. At a high-level, how does Chord know which node is responsible for a given key? What attractive properties does Chord provide?

- node's name determines keys placed there

- hash <ip address, virt. node index>

  $\Rightarrow$ m : node id

key j kept on node ~~id~~ w/ smallest id $\geq$ j



Consistent hashing

- Minimal movement of keys when nodes enter/exit
- does not need knowledge of all other nodes
- node n leaves, all n's keys re-assigned to successor

- node n joins, some of n's successor keys moved to n

- Robust to churn
  - correct w/ successor list
  - fast w/ finger ~~list~~ table

6. In more detail: What is the role of the successor list? How does a new node join a Chord ring? What is the point of the finger table?

successor: next node in chain

Keep "r" successors for fault tolerance

- if key ≤ successor, keep; otherwise forward

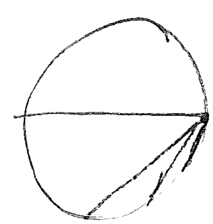
how to join? lookup key n

- that node is your successor

(notifying predecessor of new successor is more difficult)

Finger table: jump around ring, cutting distance in 1/2 each time

(get ids succeeding n by 2^{i-1} on ID circle)



Use FT to get to successor in $O(\lg N)$ lookups

7. What is server selection and why is it needed?

Mismatch between physical network layout
+ ID space

Sometimes Ids
close in ID space are far away
in physical space

When have multiple choices for routing
compromise btwn 2

8. P2P systems need to worry about malicious nodes. Can a malicious node alter data contents? What can a malicious node do? Why can't you let nodes pick their id?

+ Data fine due to SHA-1 hash

+ Can refuse to deliver data or mess up routing

- If could pick ID, control which data you serve; could selectively deny
- if in charge of all replicas, could deny

9. Why does CFS use virtual nodes? What is the danger of allowing virtual nodes?

Virtual nodes:

Some servers have better bw or more storage space

→ responsible for more keys

Must make sure 1 phys. node isn't given contiguous virt. node #

SHA-1 hash <ip, virt. node #>

→ Not contiguous + difficult to control

10. CFS stores files in fixed-sized blocks. What are the pros and cons of using fixed-sized blocks?

Big deal in comparison to PAST + Pastry (whole file)

- 8 KB blocks - ridiculously small!

+ easier to do caching

* + not as bad of contention (helps load balancing)

+ can prefetch blocks + do in parallel

- more lookups per file

11. Since nodes are free to leave the system at any time, a P2P system must take special care to ensure availability. How does CFS do this?

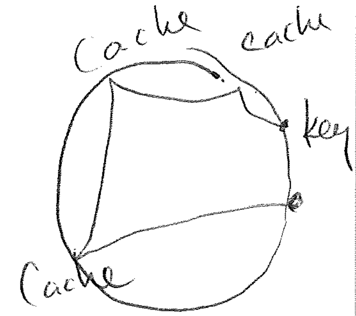
- Replicate data on k nodes
  after successor - keep it replicated

- Have $r \geq k$ in successor list

Lookup actually returns key's predecessor -
(Can get data from any of successors)
useful w/ server selection

12. Caching is used to improve performance in CFS. How is it is performed? Do you think this will work well?

- Keep copy of block along entire lookup chain



- As get closer to target, more paths converge on same nodes
  - More likely to find cached copy

(Some Versions don't put cached copy other than @ end)

Seems unlikely to follow same path when far away - LOW UTILITY??!!

Caching is only for perf. Not Avail!

13. Conclusions?

- Similar to PAST / PASTRY – same SOSP
- Paper: very preliminary work
  many follow on papers
  opts for churn
  routing algorithms

apply to more controlled environ?
(large data center w/ failures)

- distributed hash tables important
- content-addressable storage important