Section 13: Distributed Systems

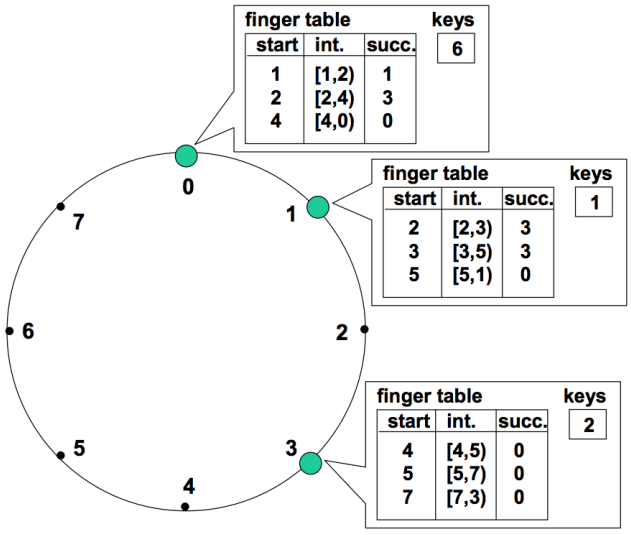
CS 162

November 27, 2020

Contents

1 Vocabulary

- **Network File System (NFS)** - A distributed file system written by Sun. NFS is based on a stateless RPC protocol. Buffers are **write behind**. Few strong consistency guarantees on parallel writes. NFS is **eventually consistent**.
- **Andrew File System (AFS)** - A distributed file system written at CMU. Full files are buffered locally upon `open`. Buffers are **write back** and only flushed on `close`.
- **Distributed Hash Table (DHT)** - A distributed hash table, or a distributed key value store, is a system which follows the semantics of a regular key value store, but in which the data is distributed over multiple machines.
- **Recursive Query** - A DHT query strategy in which requests are made to a central directory, which acts as a proxy to reroute the request to the appropriate data server. Recursive queries tend to have lower latency, and provide for an easier consistency model, but don't scale as well.
- **Iterative Query** - A DHT query strategy in which a lookup occurs to resolve a node name. The client then directly connects to the node to continue the query. Iterative queries tend to have higher latencies, and are more difficult to design for consistency, but provide more scale. Many GFS based KV Stores follow this model.
- **Consistent Hashing** - A technique for assigning a K/V Pair to a node. With consistent hashing, a new node can be added to a DHT while only moving a fraction (K/N) of the total keys. With consistent hashing Nodes are placed in the key space. A node is responsible for all the keys less than it, but greater than its predecessor. When a new node joins, it copies its necessariy data from its successor.
- **Chord** - Chord is a distributed lookup protocol for efficiently resolving the node corresponding to a key in a DHT. Chord uses a finger table which contains pointers to exponentially further nodes provide $\log(N)$ lookup time. It periodically updates the fingertable to provide for eventual consistency.



- **Replication** A strategy for fault tolerance. With replication, one or more **replicas** are responsible for trying to maintain the same state.

2 Distributed File Systems

Distributed file systems must provide the same access API as standard file systems. That access API quickly becomes non standard in the face of concurrent operations.

Compare the performance of AFS, NFS, EXT4, and EXT4 with the streaming api. Assume processes run on separate machines wherever it is applicable.

1. **open**

2. **write**

3. **read**

4. **close**

Now compare the behavior of AFS, NFS, EXT4, and EXT4 with the streaming api (with a large buffer). Assume processes run on separate machines wherever it is applicable and that buffers are only flushed when filled and upon close.

1. Process A and Process B write to a file simultaneously, then Process A closes the file, then Process B closes the file.

2. Process A increments an integer (using read and write), 1 second later, Process B increments the integer too. Both processes close the file.

3. Process A increments an integer (using read and write), 1 minute later, Process B increments the integer too. Both processes close the file.

4. Process A writes a large amount of data, then Process B writes a large amount of data. Then Process B closes the file, then Process A closes the file.

3 Distributed Key-Value Stores

a) Consider a distributed key-value store using a directory-based architecture.

Keys are 256 bytes, values are 128 MiB, each machine in the cluster has a 8 GiB/s network connection, and the client has a unlimited amount of bandwidth. The RTT between the directory and data machines is 2ms and the RTT between the client and directory/data nodes is 64ms.

i) How long would it take to execute a single GET request using a recursive query?

ii) How long would it take to execute 2048 GET requests using recursive queries?

iii) How long would it take to execute a single GET request using an iterative query?

iv) How long would it take to execute 2048 GET requests using an iterative query?

v) Now imagine our client is located in the same datacenter, and the RTT between all components is the same (this is a common assumption when modeling datacenter topology).

Briefly describe how your results would change.

vi) What are some advantages and disadvantages to using a recursive query system?

vii) What are some advantages and disadvantages to using an iterative query system?

b) **Quorum consensus:** Consider a fault-tolerant distributed key-value store where each piece of data is replicated N times. If we optimistically return from a put() call as soon as we have received acknowledgements from W replicas, how many replicas must we wait for a response from in a get() query in order to guarantee consistency?

c) In a distributed key-value store, we need some way of hashing our keys in order to roughly evenly distribute them across our servers. A simple way to do this is to assign key K to server i such that $i = hash(K)N$, where N is the number of servers we have. However, this scheme runs into an issue when N changes — for example, when expanding our cluster or when machines go down. We would have to re-shuffle all the objects in our system to new servers, flooding all of our servers with a massive amount of requests and causing disastrous slowdown. Propose a hashing scheme (just an idea is fine) that minimizes this problem.

d) Consider a distributed key value store, in which for each KV pair, that pair is stored on a single node machine, and we use iterative querying (this is essentially what we looked at in the previous DHT problem).

(a) Describe some limitations of this system. In particular, focus on bandwidth and durability.

(b) Propose some strategies for overcoming these limitations.