Section 14: Distributed KV Stores and 2PC

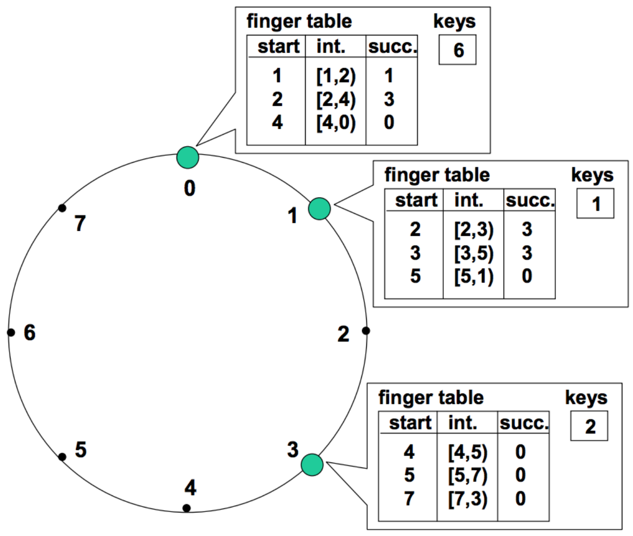
CS 162

August 10, 2020

Contents

1 Vocabulary

- **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.
- **Primary/Secondary** or **Coordinator/Worker**. A scheme for separation of responsibilities. In this scheme, there is typically a single active primary and one or more secondaries. The primary is typically responsible for being the source of truth and directing operations towards the secondaries.
- **Failover** A fault tolerance procedure invoked when a component fails. Typically this involves switching over to, or promoting a secondary.

- **2PC** - Two Phase Commit (2PC) is an algorithm that coordinates transactions between one coordinator and many workers. Transactions that change the state of the worker are considered 2PC transactions and must be logged and tracked according to the 2PC algorithm. 2PC ensures atomicity and durability by ensuring that a write happens across ALL replicas or NONE of them. The replication factor indicates how many different workers a particular entry is copied among. The sequence of message passing is as follows:

```
for every worker replica and an ACTION from the coordinator,
origin [MESSAGE] -> dest :
---
COORDINATOR [VOTE-REQUEST(ACTION)] -> WORKER
WORKER [VOTE-ABORT/COMMIT] -> COORDINATOR
COORDINATOR [GLOBAL-COMMIT/ABORT] -> WORKER
WORKER [ACK] -> COORDINATOR
```

  If at least one worker votes to abort, the coordinator sends a GLOBAL-ABORT. If all worker vote to commit, the coordinator sends GLOBAL-COMMIT. Whenever a coordinator receives a response from a worker, it may assume that the previous request has been recognized and committed to log and is therefore fault tolerant. (If the coordinator receives a VOTE, the coordinator can assume that the worker has logged the action it is voting on. If the coordinator receives an ACK for a GLOBAL-COMMIT, it can assume that action has been executed, saved, and logged such that it will remain consistent even if the worker dies and rebuilds.)

2 Distributed Key-Value Stores

1. 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.

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

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

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

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

   (e) 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.

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

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

2. **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?

3. 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 $i = \text{hash}(K) \mod N$, 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.

4. 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.

3 Two-Phase Commit

Quorum consensus is able to provide weak consistency. For this section, assume the DHT backs each node with multiple replicas. Further, assume that these machines use a two phase commit protocol to commit information in a strongly consistent manner.

1. 2PC requires a single coordinator and at least one worker. By default, all replicas can be workers. How should the coordinator be picked?

2. Briefly describe the messages that the **coordinator** will send and receive in response to a PUT. Also describe when and what would be logged.

3. Briefly describe the messages that a **worker** will send and receive.

4. Under the current model, multiple PUT queries could be sent to separate coordinator machines. Propose set of worker routines which can handle this. In particular, consider the case in which 2 coordinators receive PUT queries for **the same key but different values**. The state of the system should remain consistent.