Logistics

• HW6 will be released by tonight.
  • You should be able to solve the first question right-away.
  • You should be able to solve the first two parts of the second question after today's class.
  • You should be able to solve the remaining questions by the end of next week.
Our agenda for the next 3-4 classes

• Brief overview of key-value stores

• Distributed Hash Tables
  • Peer-to-peer protocol for efficient insertion and retrieval of key-value pairs.

• Key-value stores in the cloud
  • How to run large-scale distributed computations over key-value stores?
    • Map-Reduce Programming Abstraction
  • How to design a large-scale distributed key-value store?
    • Case-study: Facebook’s Cassandra
Our focus today

• Brief overview of key-value stores

• Distributed Hash Tables
  • Peer-to-peer protocol for efficient insertion and retrieval of key-value pairs.

• Key-value stores in the cloud
  • How to run large-scale distributed computations over key-value stores?
    • Map-Reduce Programming Abstraction
  • How to design a large-scale distributed key-value store?
    • Case-study: Facebook’s Cassandra
The Key-value Abstraction

• (Business) Key → Value
  • (twitter.com) tweet id → information about tweet
  • (amazon.com) item number → information about it
  • (kayak.com) Flight number → information about flight, e.g., availability
  • (yourbank.com) Account number → information about it
The Key-value Abstraction (2)

• It’s a dictionary data-structure.
  • Insert, lookup, and delete by key
  • E.g., hash table, binary tree

• But *distributed*. 
Isn’t that just a database?

- Yes, sort of.
- Relational Database Management Systems (RDBMSs) have been around for ages
  - e.g. MySQL is the most popular among them
- Data stored in structured tables based on a Schema
  - Each row (data item) in a table has a primary key that is unique within that table.
- Queried using SQL (Structured Query Language).
  - Supports joins.
Relational Database Example

Example SQL queries
1. SELECT zipcode
   FROM users
   WHERE name = “Bob”

2. SELECT url
   FROM blog
   WHERE id = 3

3. SELECT users.zipcode, blog.num_posts
   FROM users JOIN blog
   ON users.blog_url = blog.url
Mismatch with today’s workloads

- Data: Large and unstructured
- Lots of random reads and writes
- Sometimes write-heavy
- Foreign keys rarely needed
- Joins infrequent
Key-value/NoSQL Data Model

• NoSQL = “Not Only SQL”
• Necessary API operations: get(key) and put(key, value)

• Tables
  • Like RDBMS tables, but …
  • May be unstructured: May not have schemas
    • Some columns may be missing from some rows
  • Don’t always support joins or have foreign keys
  • Can have index tables, just like RDBMSs
Key-value/NoSQL Data Model

- Unstructured
- No schema imposed
- Columns missing from some Rows
- No foreign keys, joins may not be supported
Key-value/NoSQL Data Model

• NoSQL = “Not Only SQL”
• Necessary API operations: `get(key)` and `put(key, value)`

• Tables
  • Like RDBMS tables, but …
  • May be unstructured: May not have schemas
    • Some columns may be missing from some rows
  • Don’t always support joins or have foreign keys
  • Can have index tables, just like RDBMSs
Our focus today

- Brief overview of key-value stores

- Distributed Hash Tables
  - Peer-to-peer protocol for efficient insertion and retrieval of key-value pairs.

- Key-value stores in the cloud
  - How to run large-scale distributed computations over key-value stores?
    - Map-Reduce Programming Abstraction
  - How to design a large-scale distributed key-value store?
    - Case-study: Facebook’s Cassandra
Distributed Hash Tables (DHTs)

- Multiple protocols were proposed in early 1990s.
  - Chord, CAN, Pastry, Tapestry
  - Initial usecase: Peer-to-peer file sharing
    - key = hash of the file, value = file
  - Cloud-based distributed key-value stores reuse many techniques from these DHTs.

- Key goals:
  - Balance load uniformly across all nodes (peers).
  - Fault-tolerance
  - Efficient inserts and lookups.
Distributed Hash Tables (DHTs)

- Multiple protocols were proposed in early 1990s.
  - **Chord**, CAN, Pastry, Tapestry
  - Initial usecase: Peer-to-peer file sharing
    - key = hash of the file, value = file
  - Cloud-based distributed key-value stores reuse many techniques from these DHTs.

- Key goals:
  - Balance load uniformly across all nodes (peers).
  - Fault-tolerance
  - Efficient inserts and lookups.
Chord

• Developed at MIT by I. Stoica, D. Karger, F. Kaashoek, H. Balakrishnan, R. Morris, Berkeley and MIT

• Key properties:
  • Load balance:
    • spreads keys evenly over nodes.
  • Decentralized:
    • no node is more important than others.
  • Scalable:
    • cost of key lookup is $O(\log N)$, $N = \text{no. of nodes}$.
  • High availability:
    • automatically adjusts to new nodes joining and nodes leaving.
  • Flexible naming:
    • no constraints on the structure of keys that it looks up.
Chord: Consistent Hashing

- Uses Consistent Hashing on node’s (peer’s) address
  - SHA-1 (ip_address, port) \(\rightarrow\) 160 bit string
  - Truncated to \(m\) bits (modulo \(2^m\))
  - Called peer id (number between 0 and \(2^m - 1\))
  - Not unique but id conflicts very unlikely
  - Can then map peers to one of \(2^m\) logical points on a circle

Circle for \(m = 3\)
Chord: Consistent Hashing

- Uses Consistent Hashing on node’s (peer’s) address
  - SHA-1 (ip_address, port) → 160 bit string
  - Truncated to \( m \) bits (modulo \( 2^m \))
  - Called peer id (number between 0 and \( 2^m - 1 \))
  - Not unique but id conflicts very unlikely
  - Can then map peers to one of \( 2^m \) logical points on a circle

Where will N16 be placed on this circle?
Chord: Consistent Hashing

- Uses Consistent Hashing on node’s (peer’s) address
  - SHA-1 (ip_address, port) → 160 bit string
  - Truncated to $m$ bits (modulo $2^m$)
  - Called peer id (number between 0 and $2^m - 1$)
  - Not unique but id conflicts very unlikely
  - Can then map peers to one of $2^m$ logical points on a circle

Where will N45 be placed on this circle?
Ring of Peers: Running Example

- Say $m=7$ (128 possible points on the circle – not shown)
- 6 nodes in the system.
Mapping Keys to Nodes

- Use the same consistent hash function
  - SHA-1(key) → 160 bit string (key identifier)
    - Henceforth, we refer to SHA-1(key) as key.
  - The key-value pair stored at the key’s successor node.
  - \( \text{successor(key)} = \text{first peer with id greater than or equal to } (\text{key mod } 2^m) \)
    - Cross-over the ring when you reach the end.
      - \( 0 < 1 < 2 < 3 \ldots \ldots < 127 < 0 \) (for \( m=7 \))

- Consistent Hashing => with K keys and N peers, each peer stores \( O(K/N) \) keys. (i.e., < c.K/N, for some constant c)
Ring of Peers: Running Example

Where will the value with key 42 be stored?
Where will the value with key 42 be stored?
Where will the value with key 115 be stored?
Suppose N80 receives a request to lookup K42.

What is the value for K42?

Need to ask the successor of K42!
Performing Lookups

• Option 1: Each node is aware of (can route to) any other node in the system.
  • Need a very large routing table.
  • Poor scalability with 1000s of nodes.
  • Any node failure and join will require a necessary update at all nodes.

• Option 2: Each node is aware of only its ring successor.
  • O(N) lookup. Not very efficient.

• Chord chooses a sweet middle-ground.
Performing Lookups

• Chord chooses a sweet middle-ground.
  • Each node is aware of $m$ other nodes.
  • Maintains a finger table with $m$ entries.
  • The $i$th entry of node $n$'s finger table = $\text{successor}(n + 2^i)$
    • $i$ ranges from 0 to $m-1$
Finger Tables

Compute the finger table for N80
Finger Tables

Finger Table at N80

<table>
<thead>
<tr>
<th>i</th>
<th>ft[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>96</td>
</tr>
<tr>
<td>1</td>
<td>96</td>
</tr>
<tr>
<td>2</td>
<td>96</td>
</tr>
<tr>
<td>3</td>
<td>96</td>
</tr>
<tr>
<td>4</td>
<td>96</td>
</tr>
<tr>
<td>5</td>
<td>112</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
</tr>
</tbody>
</table>

Say $m=7$
Performing Lookups

Suppose N80 receives a request to lookup K42.

What is the value for K42?

Need to locate successor of K42!
Which nodes is N80 aware of?

Finger Table at N80

<table>
<thead>
<tr>
<th>i</th>
<th>ft[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>96</td>
</tr>
<tr>
<td>1</td>
<td>96</td>
</tr>
<tr>
<td>2</td>
<td>96</td>
</tr>
<tr>
<td>3</td>
<td>96</td>
</tr>
<tr>
<td>4</td>
<td>96</td>
</tr>
<tr>
<td>5</td>
<td>112</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
</tr>
</tbody>
</table>

Say $m=7$
Performing Lookups

Suppose N80 receives a request to lookup K42.

What is the value for K42?

Need to locate successor of K42!
Forward the query to the most promising node you know of.
Search for key \( k \) at node \( n \)

At node \( n \), if \( k \) lies in range \( (n, \text{next}(n)] \), where \( \text{next}(n) \) is \( n \)'s ring successor then \( \text{next}(n) = \text{successor}(k) \). Send query to \( \text{next}(n) \)

Else, send query for \( k \) to largest finger entry \( \leq k \)

What is the value for \( K42 \)?
Analysis

Search takes $O(\log(N))$ time

Proof Intuition:

• (intuition): at each step, distance between query and peer-with-file reduces by a factor of at least 2 (why?)

• (intuition): after $\log(N)$ forwardings, distance to key is at most $2^m/2^{\log(N)} = 2^m / N$

• Expected number of node identifiers in a range of $2^m / N$:
  • ideally one
  • $O(\log(N))$ with high probability (by properties of consistent hashing)

So using ring successors in that range will use another $O(\log(N))$ hops. Overall lookup time stays $O(\log(N))$. 
Analysis

- $O(\log(N))$ search time holds for file insertions too (in general for routing to any key)
  - “Routing” can thus be used as a building block for
    - all operations: insert, lookup, delete
- $O(\log(N))$ time true only if finger and successor entries correct
- When might these entries be wrong?
  - When you have failures
    - Next class!
MP3: Distributed Transactions

- [https://courses.grainger.illinois.edu/cs425/sp2021/mps/mp3.html](https://courses.grainger.illinois.edu/cs425/sp2021/mps/mp3.html)
- Lead TA: Dayue Bai

**Task:**
- Build a distributed transaction system that satisfies ACI properties (you do not need to handle Durability).

**Objective:**
- Think through and implement algorithms for achieving atomicity and consistency with distributed transactions (two-phase commit), concurrency control (two-phase locking / timestamped ordering), deadlock detection.
MP3: Distributed Transactions

Use this information to establish communication across servers.

A sp21-cs425-g01-01.cs.illinois.edu 1234
B sp21-cs425-g01-02.cs.illinois.edu 1234
C sp21-cs425-g01-03.cs.illinois.edu 1234
D sp21-cs425-g01-04.cs.illinois.edu 1234
E sp21-cs425-g01-05.cs.illinois.edu 1234
MP3: Distributed Transactions

![Diagram showing distributed transactions across multiple servers and a client]

Sample config file:

A sp21-cs425-g01-01.cs.illinois.edu 1234
B sp21-cs425-g01-02.cs.illinois.edu 1234
C sp21-cs425-g01-03.cs.illinois.edu 1234
D sp21-cs425-g01-04.cs.illinois.edu 1234
E sp21-cs425-g01-05.cs.illinois.edu 1234
MP3: Distributed Transactions

server A

server B

server C

server D

server E

client

Receives user input (command) from stdin.
Prints output of the command to stdout.

< BEGIN //start a new transaction
MP3: Distributed Transactions

For each transaction, client randomly chooses a server to act as coordinator. Only communicates with the coordinator.

server A

server B

server C

server D

server E

client

Receives user input (command) from stdin.

Prints output of the command to stdout.

< BEGIN //start a new transaction

> OK

< DEPOSIT A.foo 10 //deposit 10 units in account foo at branch A
MP3: Distributed Transactions

Receives user input (command) from stdin.
Prints output of the command to stdout.

< BEGIN //start a new transaction
> OK
< DEPOSIT A.foo 10 //deposit 10 units in account foo at branch A
> OK
MP3: Distributed Transactions

client

server A  server B  server C  server D  server E

Receives user input (command) from stdin.
Prints output of the command to stdout.

< BEGIN //start a new transaction
> OK
< DEPOSIT A.foo 10 //deposit 10 units in account foo at branch A
> OK

Other possible commands: WITHDRAW and BALANCE (only applicable if the account exists)
MP3: Distributed Transactions

User enters COMMIT or ABORT to end the transaction.

A server may also choose to ABORT a transaction (e.g. if consistency violated, or if needed for concurrency control).

Changes made by one transaction visible to others only after it successful commits.
MP3: Distributed Transactions

Required properties:
• Atomicity:
  • all servers commit the entire transaction, or all rollback the entire transaction.
• Consistency:
  • cannot withdraw from or read balance of a non-existent account.
  • a transaction cannot result in a negative account balance.
MP3: Distributed Transactions

Receives user input (command) from stdin.
Prints output of the command to stdout.

Required properties:
• Isolation:
  • multiple clients may concurrently issue commands on the object.
  • Must provide serial equivalence.
• Deadlock avoidance.
**MP3: Distributed Transactions**

- Due on Friday, May 5th.
  - Allowed to submit up to 50 hours late, but with 2% penalty for every late hour (rounded up).

- Read the specification fully and carefully.
  - Required semantics discussed more completely there.

- Start early!