Distributed Data Management
Summer Semester 2013
TU Kaiserslautern

Dr.-Ing. Sebastian Michel

smichel@mmci.uni-saarland.de
Lecture 7

NOSQL: DATA REPLICATION & SYNCHRONIZATION
Brief Announcements

• **Exams:** Created doodle poll (anonymous). Just put your information (name/email) in ONE free slot.

• We will send an email with the poll URL soon. Then, first come first serve.

• **No lecture on June 27**
Replication

• Consider key/value store
• Each object is replicated N times (have seen before). Now we look at how.

• How to make sure updates are eventually arriving at replica holding nodes?
Recap: Consistent Hashing

- “Random” placement of nodes and keys in cyclic identifier space.
- Nodes come and go: only neighbors are affected.
Routing in a Consistent Hashing “Structure”

• A client does not know server for key, but some (any) other server

• Naïve routing:
  – Each node knows its neighbor
  – Send message to nearest neighbor
  – Getting closer to target node with each hop
  – But $O(n)$ cost!
Routing with Logarithmic Cost

- Each node keeps a lookup table (also called finger table)
- At exponentially increasing distances.
- Periodically refreshed.
- Routing in $O(\log(n))$

<table>
<thead>
<tr>
<th>Key</th>
<th>Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>key(this)+1</td>
<td>192.168.434.12</td>
</tr>
<tr>
<td>key(this)+2</td>
<td>...</td>
</tr>
<tr>
<td>key(this)+4</td>
<td></td>
</tr>
<tr>
<td>key(this)+8</td>
<td></td>
</tr>
<tr>
<td>key(this)+16</td>
<td></td>
</tr>
<tr>
<td>key(this)+32</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Finger Table Routing Example

**Table:**

<table>
<thead>
<tr>
<th>$p_{51} + 1$</th>
<th>$p_{56}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_{51} + 2$</td>
<td>$p_{56}$</td>
</tr>
<tr>
<td>$p_{51} + 4$</td>
<td>$p_{56}$</td>
</tr>
<tr>
<td>$p_{51} + 8$</td>
<td>$p_{1}$</td>
</tr>
<tr>
<td>$p_{51} + 16$</td>
<td>$p_{8}$</td>
</tr>
<tr>
<td>$p_{51} + 32$</td>
<td>$p_{21}$</td>
</tr>
</tbody>
</table>

**Table:**

<table>
<thead>
<tr>
<th>$p_{42} + 1$</th>
<th>$p_{48}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_{42} + 2$</td>
<td>$p_{48}$</td>
</tr>
<tr>
<td>$p_{42} + 4$</td>
<td>$p_{48}$</td>
</tr>
<tr>
<td>$p_{42} + 8$</td>
<td>$p_{51}$</td>
</tr>
<tr>
<td>$p_{42} + 16$</td>
<td>$p_{1}$</td>
</tr>
<tr>
<td>$p_{42} + 32$</td>
<td>$p_{14}$</td>
</tr>
</tbody>
</table>

**Table:**

<table>
<thead>
<tr>
<th>$p_{8}$</th>
<th>$p_{14}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_{8} + 1$</td>
<td>$p_{14}$</td>
</tr>
<tr>
<td>$p_{8} + 2$</td>
<td>$p_{14}$</td>
</tr>
<tr>
<td>$p_{8} + 4$</td>
<td>$p_{14}$</td>
</tr>
<tr>
<td>$p_{8} + 8$</td>
<td>$p_{21}$</td>
</tr>
<tr>
<td>$p_{8} + 16$</td>
<td>$p_{32}$</td>
</tr>
<tr>
<td>$p_{8} + 32$</td>
<td>$p_{42}$</td>
</tr>
</tbody>
</table>

**Diagram:**

- **Chord Ring**
- **Fingertable $p_{51}$**
- **Fingertable $p_{42}$**
- **Fingertable $p_{8}$**
- **Lookup(54)**
Node Joining Ring

$p_{42}$ lookup(42)

$p_{42}$ sets succ pointer

moving keys

$p_{38}$ updates succ pointer
Consistent Hashing in Dynamo

• Global view of partitioning following the principles of consistent hashing

• No routing tables, no multi-hop routing (reason, network roundtrips is too expensive for low latency) (check SLA=Service Level Agreements, e.g., 300ms for 99.9%)

• Instead: dissemination of full network information, using gossiping as information dissemination (will see later) => O(1) lookup cost
Replication in Amazon’s Dynamo

Replica holders are physically distinct nodes (because of virtual nodes).

<table>
<thead>
<tr>
<th>Key</th>
<th>Node</th>
<th>Replica</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>A</td>
<td>B, C</td>
</tr>
<tr>
<td>12</td>
<td>B</td>
<td>C, D</td>
</tr>
<tr>
<td>19</td>
<td>B</td>
<td>C, D</td>
</tr>
<tr>
<td>20</td>
<td>C</td>
<td>D, E</td>
</tr>
<tr>
<td>37</td>
<td>D</td>
<td>E, F</td>
</tr>
<tr>
<td>40</td>
<td>E</td>
<td>F, A</td>
</tr>
<tr>
<td>54</td>
<td>F</td>
<td>A, B</td>
</tr>
</tbody>
</table>

Replicas are stored at X (here 2) successors of node that “owns” the key.
Hinted Handoff

• What if a node for a key is not available?

• Store data at other node, coordinator or neighbor. With hint that it is for the (currently) unavailable node.
Hinted Handoff (Cont’d)

• Problem: Hinted Handoff information can get lost if the holding node is unavailable.

• Requires protocol that fixes such inconsistencies.

• Each node stores a set of entries of the form <key, value, version>

• According to, here, ranges on the “ring”, but protocols we see are independent on that.
Synchronization Process

• Given N nodes (replicas)
• Each of them might or might not have the recent value of an object

• Communication between nodes has to ensure consistent view on data (replicas)
Deterministic Solution

• Node that gets new information sends the information to the N-1 other nodes. (Also called direct mail).

• Pros and Cons?
• Very efficient, no duplicate messages that waste network bandwidth or CPU time.
• But what if a node fails?
Epidemic Algorithms

• **Anti-entropy:** Information is constantly exchanged with randomly selected node. Items to be exchanged are always the current versions items stored in the nodes. Do that continuously.

• **Rumor spreading:** Information is exchanged with randomly chosen nodes, multiple rounds, then stop. With high probability, data is consistently replicated afterwards.
Rumor Spreading

• Think: Spreading rumors between people, say, via phone calls

• Two issues: Understanding how rumor spreads (in social networks, e.g.,) or how to devise algorithms that behave similarly
Variants of Gossiping

• **Push**: Holder of new information actively distributes it.

• **Pull**: People actively call to obtain news.

What are the strong and weak characteristics of both strategies?
Push

initial

round 1

round 2

round 3

round 4

Shown only relevant pulls for illustration.

= old version / no rumor

= new version / knows rumor
Pull

initial  

round 1
Shown only relevant pulls for illustration.
Push-Pull

• Combination of push and pull
• Also works in rounds.

• In each round:
  – each node contacts a random neighbor
  – if one of the two has the rumor it tells the other
    • push: caller sends rumor
    • pull: caller receives (learns) rumor
Behavior

• Rumor spreading in case of complete graphs, random graphs or hypercube graphs:
  in $O(\log n)$ rounds all nodes know the rumor with high probability (w.h.p.)

Also robust to failures: if communication links fail with certain probability $f<1$ then, e.g., $O(1/(1-f))$ more time needed

Anti-Entropy as Secondary Protocol

• Demers et al. put Anti-Entropy in the role of being used after “direct mail” or rumor spreading protocols.
• To fix missing information due to unavailable nodes or
• in case rumor spreading did not receive 100% of all nodes (as it comes with only a “with high probability” guarantee)

Anti-Entropy as Secondary Protocol (2)

• If used in the case of majority of nodes are in sync already, then

• Pull or pull-push is much better suitable then push only. Why?

Say $p_i$ is probability that a node is not informed, then in next round

for pull: $p_{i+1} = (p_i)^2$

for push: $p_{i+1} = p_i \times (1 - \frac{1}{n})^n(1-p_i)$

source: Demers et al.
Optimizing Data Exchange/Comparison

• Points before addresses the protocols for data exchange between two nodes.
• In each such process, lots of data is required to be sent/processed.

• Large potential for optimization through compression (signatures).
• First shot: use checksums (e.g., MD5 or SHA-1) of data
• If checksum is the same, data is precisely the same (w.h.p.)
Merkle Trees

• Hash Trees (invented 1979 by R. Merkle)
• Parent node is hash of its children
• Used in distributed systems for checking consistency of data
• Allows hierarchical checking
Comparing Merkle Trees

• Start at root.
• If the same hash, then stop.
• Otherwise: compare corresponding nodes in levels. For nodes with diff. hash: go down to children, etc.
• Eventually: Found different data (leaves) => exchange them

Distributed Data Management, SoSe 2013, S. Michel
Merkle Trees in Dynamo

• Each node maintains a separate Merkle tree for each key range (as we have multiple due to virtual nodes!)

• Two nodes compare Merkle trees of the ranges they have in common, as described before.
Partitioning / Replication & Dynamics

• Have seen consistent hashing
• Now, slight variations for (said) better performance (again, in Dynamo)

• Dynamics: new nodes cause key ranges of nodes to change.
• Merkle trees need to be recomputed
• Data for “moving” ranges gathered and transferred.
Traditional Consistent Hashing

- S*T nodes are placed randomly (S=number of real nodes, T=virtual instances per node, called also Tokens in*)

- Range between them defines partitions

- N copies of partitions in N-1 successors of node that hashing tells to be responsible

not possible to add nodes without affecting data partitioning

Random Placement with equal Sized Partitions

• Have Q equal sized partitions (Q >> T*S), where

• Nodes are (as before) placed randomly.

• Partition is assigned to N nodes that follow (successors) the end of the partition.

Decoupling of partitioning and partition placement Partition bounds don’t change. Efficient maintenance.
Q/S Virtual Nodes for each Node

• Q/S virtual nodes per node (S=number of nodes in system)
• I.e., one partition per virtual node
• When node enters: steals positions from existing ones
• At leave: gives back
• Such that property remains (means: extra work to do!)

Best load balancing among discussed schemes.
Literature

• Richard M. Karp, Christian Schindelhauer, Scott Shenker, Berthold Vöcking: Randomized Rumor Spreading. FOCS 2000: 565-574
• Ralph C. Merkle: A Digital Signature Based on a Conventional Encryption Function. CRYPTO 1987: 369-378
• http://www.allthingsdistributed.com/2007/10/amazons_dynamo.html