Distributed Data Management

Summer Semester 2015
TU Kaiserslautern

Prof. Dr.-Ing. Sebastian Michel
Databases and Information Systems Group (AG DBIS)

http://dbis.informatik.uni-kl.de/
Recap: Consistent Hashing
“Consistent Hashing” in Amazon 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) => then O(1) lookup cost
Replication in Amazon’s Dynamo

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

Replica holders are **physically distinct nodes** (because of virtual nodes).
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 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 now 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?
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 (we, here, will look at algorithms)
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 push operations 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)

- Assume case of having majority of nodes that are in sync already and have the same latest version

- What is the method of choice, push or pull?

- Latest version (=“knows rumor”)
- Not latest version (=“does not know rumor”)

Distributed Data Management, SoSe 2015, S. Michel
Anti-Entropy as Secondary Protocol (3)

• Then: Pull or pull-push is much better suitable then push only.

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

source: Demers et al.
Optimizing Data Exchange/Comparison

• Points before addresses the protocols for data exchange between two nodes.
• In each such process, potentially 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 (almost certainly)
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 different hash: go down to children,* etc.

• Eventually: Found different data (leaves) => exchange them
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 (here =3) 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** ($Q \gg T*S$), 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** \( (S=\text{number of nodes in system}) \)
- i.e., **one partition per virtual node + replication**
- 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
• http://highlyscalable.wordpress.com/2012/09/18/distributed-algorithms-in-nosql-databases/
Summary NoSQL Part

• Walked through core characteristics of fault-tolerant (replicated) distributed data stores.
• Started with simple replica management and state machine replication. t-fault tolerance. Different failure models.
• Paxos for consensus. Logical clocks and vector clocks for bringing order to “events” in a distributed system.
• CAP theorem, BASE, consistency models.
• Data placement (consistent hashing) and synchronization methods (rumor spreading).