PART 2. LARGE SCALE DATA STORAGE SYSTEMS
NO SQL DATA STORAGE

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs435

FAQs
• Term project final report
  • Preparation guide is available at:
    • http://www.cs.colostate.edu/~cs435/TP2.html
• A Successful final project must demonstrate:
  • Relevance
  • Completeness
  • Challenge

Analytics Tips IV [1/4]
• Voter Party Registration (Scott Fortmann-Roe)
  • We want to predict voter registration using wealth and religiousness as predictors

Analytics Tips IV [2/4]
• k-Nearest Neighbors algorithm
  • If the nearest voter to him (in terms of wealth and religiousness) is a Democrat, s/he will also be predicted to be a Democrat
  • The right figure shows the nearest neighborhoods for each of the original voters

Analytics Tips IV [3/4]
• k-Nearest Neighbors algorithm
  • Nearest neighbor predictions for new data

Analytics Tips IV [4/4]
• Error due to Bias
  • the difference between the expected (or average) prediction of our model and the correct value which we are trying to predict
• Error due to Variance
  • the variability of a model prediction for a given data point
• E.g. Presidential election survey
  • Selecting survey participants from a phonebook is a source of bias; a small sample size is a source of variance; minimizing total model error relies on the balancing of bias and variance errors.
Today’s topics

• No SQL storage

Part 2. Large scale data storage system

NoSQL Storage

NoSQL databases

• Basic Idea
  • Operates without a schema
  • Allows users to add fields without having to define any changes in structure first
  • Useful when dealing with nonuniform data and custom fields
  • Stands for “Not Only SQL”

• Handles data access with size and performance that demand a cluster

• Improves the productivity of application development by using a more convenient data interaction style

Polyglot persistence

• Using different data stores in different circumstances
  • Without picking a particular database for all situations

• Most organizations have a mix of data storage technologies for different circumstances
Key-Value Store

- Simple hash table
  - All access to the storage is via primary key
    - Get the value for the key
    - Put a value for the key
    - Delete a key
    - Add a key
  - "value" is stored as a blob
    - Without caring or knowing what's inside
    - Application is responsible for understanding data

This material is built based on,


Overview of Dynamo

- Partitions and replicates data using consistent hashing
- Tracks object version to provide consistency
- Uses quorum-like technique to ensure consistency among replicas
- Uses a decentralized synchronization protocol
  - Storage nodes can be added and removed from Dynamo without any manual partitioning or redistribution
  - Gossip based distributed failure detection and membership protocol

System Assumptions (1/2)

- Query model
  - Simple read and write operations to a data item
  - Uniquely identified by a key
  - State is stored as binary objects (i.e. blobs) identified by unique keys
  - Usually less than 1MB
  - No operations span multiple data items
  - No need for relational schema
- ACID/CAP Properties
  - Dynamo targets applications that operate with weaker consistency if this results in high availability
  - Dynamo does not provide any isolation guarantees

System Assumptions (2/2)

- Efficiency
  - The system needs to function on a commodity hardware infrastructure
  - Stringent latency requirements

Summary of techniques

<table>
<thead>
<tr>
<th>Problem</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Replication</td>
<td>Consistent hashing</td>
</tr>
<tr>
<td>(2) High Availability</td>
<td>Vector clocks with reconciliation during reads</td>
</tr>
<tr>
<td>(3) Handling Temporary Failures</td>
<td>Snappy-Quorum and hinted handoff</td>
</tr>
<tr>
<td>(4) Recovering from Permanent Failures</td>
<td>Anti-entropy using Merkle tree</td>
</tr>
<tr>
<td>(5) Membership and Failures Detection</td>
<td>Gossip-based membership protocol and failure detection</td>
</tr>
</tbody>
</table>
Partitioning algorithm

- Dynamically partitions the data over the set of nodes
- Distributes the load across multiple storage hosts
- Dynamo uses consistent hashing

Non-consistent hashing vs. consistent hashing

- When a hash table is resized
  - Non-consistent hashing algorithm requires to re-hash complete table
  - Consistent hashing algorithm requires only partial records of the table

Consistent hashing (1/3)

Identifier circle with $m = 3$

Consistent hash function assigns each node and key an $m$-bit identifier using a hashing function

Hashing value of an IP address

$m$-bit Identifier: 2

$m$ has to be big enough to make the probability of two nodes or keys hashing to the same identifier negligible

Consistent hashing (2/3)

Machine $B$ is the successor node of $h(key) = 1$

successor(1) = 1

Machine $B$ is the successor node of $h(key) = 1$

successor(1) = 1

Key with the $h(key)$ of 2 will be stored in machine $C$: the first node whose Identifier is equal to or follows 2 is the node $C$.

successor(2) = 5

Key with the $h(key)$ of 3 will be stored in machine $C$: the first node whose Identifier is equal to or follows 3 is the node $C$.

successor(3) = 5
**Consistent hashing (3/3)**

- If machine C leaves circle, Successor(5) will point to A
- If machine N joins circle, Successor(2) will point to N

---

**Scalable Key location**

- In consistent hashing:
  - Each node need only be aware of its successor node on the circle
  - Queries can be passed around the circle via these successor pointers until it finds resource
- What is the disadvantage of this scheme?
  - It may require traversing all N nodes to find the appropriate mapping

---

**Dynamo’s partitioning**

- Inspired by Consistent Hashing and Chord
  - Zero-hop DHT
- When a node starts for the first time
  - Chooses its set of tokens (virtual nodes in the consistent hash space)
  - Maps nodes to their respective token sets
  - Stores both tokens and nodes onto disk
- Repeated reconciliation of the membership change
  - Partitioning and placement information are propagated via the gossip-based protocol
  - Token ranges handled by its peers
  - Direct forwarding of read/write operations are possible

---

**Replication (1/3)**

- Dynamo replicates its data on multiple hosts
  - Each data item is replicated at R hosts, where R is a parameter configured “per-instance”
- Each key k is assigned to a coordinator node
  - The coordinator is managing the replication of the data items that fall within its range
  - Stores at the R-1 clockwise successor nodes in the ring
  - Each node is responsible for the region of the ring between it and its Rth predecessor
**Replication (2/3)**

- Machine D will store the key(s)
- Original range: {4, 5} = [3, 5)
- Replication of node C's range: {2,3} = [1,3)
- Replication of node B's range: {1} = [0,1)

**Replication (3/3)**

- **Preference list**
  - The list of nodes (original and replications) that is responsible for storing a particular key
- **If there are node failures**
  - Preference list contains R nodes
  - R is the replication factor
- **Virtual nodes can reduce actual number of machines in "R nodes"**
  - The preference list for a key is constructed only by distinct physical nodes

**Data Versioning**

- Dynamo provides **eventual consistency**
  - Allows for updates to be propagated to all replicas asynchronously
  - A put call may return to its caller before the update has been applied to all the replicas

**“Add to Cart” example**

- The shopping cart application requires that an "Add to Cart" operation can never be forgotten or rejected
- "Write" operation
- If the most recent cart is not available and user makes changes to an old version of the cart
  - Still the change should be meaningful and preserved
- "add to cart" and "delete item from cart" should be translated into put operation to Dynamo
- The divergent versions are reconciled later

**Maintaining vector clock**

- Dynamo treats the result of each modification as a new and immutable version of the data
  - Multiple version of data can be in the system
- **Version branching**
  - Due to the failure(s) in node(s), there are conflicting versions of an object
- **Merging**
  - Collapses multiple branches of data evolution back into one
  - Semantic reconciliation
  - E.g., merging different versions of shopping cart and preserving all of the items those client put into the cart

**Vector clocks**

- Used to capture **causality between different versions of same object**
  - Two versions of object are on parallel branches or have a causal ordering
- **Vector clock**
  - A list of (node, counter) pairs
  - One vector clock is associated with every version of every object
Definition of the vector clocks

- \( \text{VC}(x) \) denotes the vector clock of event \( x \)
- \( \text{VC}(z), \) denotes the component of that clock for process \( z \)

\[
\text{VC}(x) < \text{VC}(y) \iff \\
\forall z [\text{VC}(x)_z < \text{VC}(y)_z]
\]

- \( x \rightarrow y \) denotes that event \( x \) happens before event \( y \)
- If \( x \rightarrow y \), then \( \text{VC}(x) < \text{VC}(y) \)

Examples (with answers)

- \( \text{VC}(D1) = ([\text{Sx}, 3], [\text{Sy}, 2], [\text{Sz}, 2], [\text{Sq}, 2]) \)
- \( \text{VC}(D2) = ([\text{Sx}, 3], [\text{Sy}, 2], [\text{Sz}, 2], [\text{Sq}, 1]) \)
- \( \text{VC}(D3) = ([\text{Sx}, 3], [\text{Sy}, 2], [\text{Sz}, 1]) \)
- \( \text{VC}(D4) = ([\text{Sx}, 3], [\text{Sy}, 2], [\text{Sq}, 2], [\text{Sq}, 1]) \)

- Is \( \text{VC}(D1) > \text{VC}(D2)? \) YES
- Is \( \text{VC}(D1) > \text{VC}(D2)? \) NO
- Is \( \text{VC}(D4) > \text{VC}(D2)? \) YES

\[
\text{VC}(x) < \text{VC}(y) \iff \\
\forall z [\text{VC}(x)_z < \text{VC}(y)_z]
\]

Properties of the vector clocks

- If \( \text{VC}(a) < \text{VC}(b) \), then \( a \rightarrow b \)

- Antisymmetry:
  - If \( \text{VC}(a) < \text{VC}(b) \) then NOT (\( \text{VC}(b) < \text{VC}(a) \))

- Transitivity
  - If \( \text{VC}(a) < \text{VC}(b) \) and \( \text{VC}(b) < \text{VC}(c) \), then \( \text{VC}(a) < \text{VC}(c) \)

Examples

- \( \text{VC}(D1) = ([\text{Sx}, 3], [\text{Sy}, 2], [\text{Sz}, 2], [\text{Sq}, 2]) \)
- \( \text{VC}(D2) = ([\text{Sx}, 3], [\text{Sy}, 2], [\text{Sz}, 2], [\text{Sq}, 1]) \)
- \( \text{VC}(D3) = ([\text{Sx}, 3], [\text{Sy}, 2], [\text{Sz}, 1]) \)
- \( \text{VC}(D4) = ([\text{Sx}, 3], [\text{Sy}, 2], [\text{Sq}, 2], [\text{Sq}, 1]) \)

- Is \( \text{VC}(D1) > \text{VC}(D2)? \) YES
- Is \( \text{VC}(D1) > \text{VC}(D2)? \) YES
- Is \( \text{VC}(D4) > \text{VC}(D2)? \) YES

\[
\text{VC}(x) < \text{VC}(y) \iff \\
\forall z [\text{VC}(x)_z < \text{VC}(y)_z]
\]

Execution of \textit{get} and \textit{put} operations

- Users can send the operations to any storage node in Dynamo

- Coordinator
  - A node handling a read or write operation
  - The top \( N \) nodes in the preference list

- Client can select a coordinator
  - Route its request through a generic load balancer
  - Use a partition-aware client library
  - Directly access the coordinators
Using quorum-like system

- **R**: Minimum number of nodes that must participate in a successful read operation
- **W**: Minimum number of nodes that must participate in a successful write operation
- Setting R and W for the given replication factor of N:
  - \( R + W > N \)
  - \( W > N/2 \)
- The latency of a get (or put) operation is dictated by the slowest one of the R (or W) replicas
  - R and W are configured to be less than N

What if W is 1?

- Applications that need the highest level of availability can set W as 1
- Under Amazon’s model
  - A write is accepted as long as a single node in the system has durably written the key to its local store
  - A write request is rejected, only if all nodes in the system are unavailable

put request

- Coordinator node
  1. Generates the vector clock
     - For the new version
  2. Writes the new version locally
  3. Sends the new version to the N highest-ranked reachable nodes
     - Along with the new vector clock

get request

- The coordinator requests all existing versions of data for that key from the N highest-ranked reachable nodes
  - In the preference list
  - Waits for R responses
    - Here, R is the read quorum
    - If multiple versions of the data are collected
      - Returns all the versions it deems to be causally unrelated
  - The reconciled version superseding the current versions is written back

Sloppy quorum

- All read and write operations are performed on the first N healthy nodes from the preference list
  - May not always be the first N nodes on the hashing ring
- Hinted handoff
  - If a node is temporarily unavailable, data is propagated to the next node in the ring
  - Metadata contains information about the originally intended node
  - Stored in a separate local database and scanned periodically
  - Upon detecting that the original node is recovered,
    - A data delivery attempt will be made
    - Once the transfer succeeds, the data at the temporary node will be removed

Part 2. Large scale data storage system

NoSQL Storage: 1. Key-Value Stores (Dynamo)
   (1) Partitioning
   (2) High Availability for writes
   (3) Handling temporary failures
   (4) Recovering from permanent failures
   (5) Membership and failure detection
The data will be sent to the node D

If C is temporarily down

This data contains a hint in its metadata—node where it was supposed to be stored

After the recovery, D will send data to C

Then, it will remove the information.