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

• Voter Party Registration (Scott Fortmann-Roe)
  • We want to predict voter registration using wealth and religiousness as predictors

• 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

• 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

Find a needle in a Haystack

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 a 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

---

**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>Partitioning</td>
<td>Consistent hashing</td>
</tr>
<tr>
<td>High Availability for writes</td>
<td>Vector clocks with reconciliation during reads</td>
</tr>
<tr>
<td>Handling temporary failures</td>
<td>Snapshots and hinted handoff</td>
</tr>
<tr>
<td>Recovering from permanent failures</td>
<td>Anti-entropy using Merkle tree</td>
</tr>
<tr>
<td>Membership and failure detection</td>
<td>Gossip-based membership protocol and failure detection</td>
</tr>
</tbody>
</table>
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

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 IP address
\( m \)-bit Identifier: 2
\( m \) identifiers
\( m \) has to be big enough to make the probability of two nodes or keys hashing to the same identifier negligible

Consistent hashing (2/3)
Key with the \( h(\text{key}) = 3 \) will be stored in machine successor(3) = 5
Key with the \( h(\text{key}) = 2 \) will be stored in machine C: the first node whose Identifier is equal to or follows 2 is the node C.
next successor (1) = 1
Machine B is the successor node of \( h(\text{key}) = 1 \) next successor (2) = 5
Key with the \( h(\text{key}) = 5 \) will be stored in machine C: the node C.
Key with the \( h(\text{key}) = 3 \) will be stored in machine B.
next successor (3) = 9
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

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
Replication (2/3)

- Machine D will store the key:
  - Original range: (4, 5) = [3, 5)
  - Replication of the node C's range: (2, 3) = [1, 3)
  - Replication of the 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 more than \( R \) 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 versions 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 \( (\text{node}, \text{counter}) \) pairs
  - One vector clock is associated with every version of every object
Definition of the vector clocks

- VC(x) denotes the vector clock of event x
- VC(x)\_z denotes the component of that clock for process z
- x \rightarrow y denotes that event x happens before event y
- If x \rightarrow y, then VC(x) < VC(y)

\[ VC(x) < VC(y) \iff \forall z [VC(x)_z \leq VC(y)_z] \land \exists z' [VC(x)_z < VC(y)_z] \]

Examples

- VC(D1)=([Sx,1], [Sy,2], [Sz,2], [Sq,2])
- VC(D2)=([Sx,1], [Sy,2], [Sz,2], [Sq,1])
- VC(D3)=([Sx,1], [Sy,2], [Sq,1])
- VC(D4)=([Sx,1], [Sz,2], [Sq,1])

- Is VC(D1) > VC(D2)?  YES
- Is VC(D3) > VC(D2)?  NO
- Is VC(D4) > VC(D2)?  YES

Examples (with answers)

- VC(D1)=([Sx,1], [Sy,2], [Sz,2], [Sq,2])
- VC(D2)=([Sx,1], [Sy,2], [Sz,2], [Sq,1])
- VC(D3)=([Sx,1], [Sy,2], [Sq,1])
- VC(D4)=([Sx,1], [Sz,2], [Sq,1])

- Is VC(D1) > VC(D2)?  YES
- Is VC(D3) > VC(D2)?  NO
- Is VC(D4) > VC(D2)?  YES

Properties of the vector clocks

- If VC(a) < VC(b), then a \rightarrow b
- Antisymmetry:
  - If VC(a) < VC(b) then NOT (VC(b) < VC(a))
- Transitivity
  - If VC(a) < VC(b) and VC(b) < VC(c), then VC(a) < VC(c)