PART 2. LARGE SCALE DATA STORAGE SYSTEMS
NO SQL DATA STORAGE
WITH A CASE STUDY OF AMAZON’S DYNAMO

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

FAQs
• Term project presentations
  • 12/9 (team 1-6), 12/11 (team 6-12), 12/13 (team 13-16)
  • Please attend at least 2 presentation sessions and ask questions or provide comments
    • Participation score (attendance + question)
  • 12 minutes (including transition time)/team
  • Submit your slides (No PDF!) on canvas

• Final Report Grading criteria (12 points)
  • Relevance: Is this project using sufficient Big Data technologies? (0~4)
  • Completeness: Does this project present all the required components in depth? (0~4)
  • Challenge: Does this project demonstrate an adequate level of challenge? (0~4)
Today’s topics

• No SQL 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
NoSQL Storage Data Model: (1) Key-Value Store

- Imagine a 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

NoSQL Storage Data Model: (2) Document Storage Model

- Documents
  - Self-describing
  - Data structure
    - Maps, collections, tree, and scalar values
    - Stores documents in the value part of the key-value store

- MongoDB, CouchDB, OrientDB, RavenDB, etc.

- Users can query the data inside the document
  - without having to retrieve the whole document
NoSQL Storage Data Model: (3) Column-Family Stores

- Cassandra, Hbase, Hypertable, and Amazon SimpleDB
- Stores data in **column family as rows**
  - Have many columns associated with a row key

- Column families
  - Groups of related data that is often accessed together

Part 2. Large scale data storage system

NoSQL Storage: Key-Value Stores

Dynamo
This material is built based on,


What Amazon needs (1/2)

- Amazon’s architecture
  - A highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services

- Storage technologies that are always available

- Customer should be able to view and add items to the shopping cart even if:
  - The disks are failing
  - Network routes are flapping or,
  - Data centers are being destroyed by tornados
If you design a data storage system for,

• Amazon.com to store transaction data for the shopping cart management, how would you prioritize properties: Consistency, Availability, or Partition tolerance? And Why?

What Amazon needs

• Highly available system with failure resilience
  • Small but significant number of servers
  • Network components

• Failure handling should not impact availability or performance
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
- DynamoDB supports document storage features as well
  - We will focus on the original Dynamo’s key-value storage features and architecture

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 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) Partitioning</td>
<td>Consistent Hashing</td>
</tr>
<tr>
<td>(2) High Availability for writes</td>
<td>Vector clocks with reconciliation during reads</td>
</tr>
<tr>
<td>(3) Handling temporary failures</td>
<td>Sloppy 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 failure detection</td>
<td>Gossip-based membership protocol and failure detection</td>
</tr>
</tbody>
</table>
Part 2. Large scale data storage system

NoSQL Storage: 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

Data partitioning algorithm

• Dynamically partitions the data over the set of nodes
• Distributes the load across multiple storage hosts
• Dynamo uses consistency hashing
  • zero-hop DHT
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

$m$-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

Hashing value of IP address
Consistent hashing (2/3)

Key 2 will be stored in machine C
successor(2) = 5

Key 3 will be stored in machine C
successor(3) = 5

Consistent hashing assigns keys to nodes: Key k will be assigned to the first node whose identifier is equal to or follows k in the identifier space.

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

Consistent hashing (3/3)

If machine C leaves circle, Successor(5) will points to A

If machine N joins circle, successor(2) will points to N

New node 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 until falling into the first bucket it contains the specified key

• What is the disadvantage of this scheme?

• It may require traversing all N nodes to find the appropriate mapping
Dynamo’s partitioning

- **Zero-hop DHT**
  - Each node maintains entire mapping list of key range and the storage node
  - Inspired by *Consistent Hashing and Chord*

- **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**
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 (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 \( R-I \) clockwise successor nodes in the ring
    - Coordinator node and and (R-1) consecutive successor nodes
  - Each node is responsible for the region of the ring between it and its \( R^{th} \) predecessor

Replication (2/3)

Machine D will store the keys, \([0,1),[1,3), and [3,5]\)
Replication (2/3)

Machine D will store the keys, 
(0,1), (1,3), and (3,5)

Machine C will store the keys, 
(1,3), (0,1), and (5,0)
Replication (2/3)

Machine B will store the keys, [0,1), [1,3), and [3,5)

Machine C will store the keys, [1,3), [0,1), and [5,0)

Machine D will store the keys, [0,1), [1,3), and [3,5)
Replication (3/3)

• **Preference list**
  - The list of nodes that is responsible for storing a particular key

• If there are **node failures**
  - Preference list contains more than $R$ nodes

• 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

- 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

• $VC(x)$ denotes the vector clock of event $x$
• $VC(x)_z$ denotes the component of that clock for process $z$

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

• $x \rightarrow y$ denotes that event $x$ happens before event $y$
  • If $x \rightarrow y$, then $VC(x) < VC(y)$
Examples

• $VC(D1) = ([Sx, 3], [Sy, 2], [Sz, 2], [Sq, 2])$
• $VC(D2) = ([Sx, 3], [Sy, 2], [Sz, 2], [Sq, 1])$
• $VC(D3) = ([Sx, 3], [Sy, 2], [Sq, 1])$
• $VC(D4) = ([Sx, 3], [Sy, 3], [Sz, 2], [Sq, 1])$

Is $VC(D1) > VC(D2)$?  
Is $VC(D3) > VC(D2)$?  
Is $VC(D4) > VC(D2)$?

$VC(x) < VC(y) \iff \\
\forall z [VC(x) \leq VC(y) \land \exists z' [VC(x) < VC(y)]$
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 $VC(a) < VC(c)$
Execution of get and 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