CS 133: Databases

Fall 2018
Lec 25 – 12/06
NoSQL

Prof. Beth Trushkowsky

Final Exam Logistics

• Final exam take-home
  – Available: Friday December 14th, 4:00pm in Olin
  – Due: Monday December 17th, 5:15pm

• Same resources as midterm
  – Except this time, two note sheets allowed (can re-use your own from midterm)

Goals for Today

• Discuss data replication in distributed DBMSs

• Understand the motivation and goals for “NoSQL” data management systems

• Reason about the key concepts, techniques, and tradeoffs for NoSQL systems
  – Touch on a couple specific NoSQL systems
    (Dynamo, MongoDB, Cassandra)

Some References Used

• Ten Rules for Scalable Performance in “Simple Operation” Datastores
  – Communications of the ACM 2011
  – Stonebraker and Cattell

• Scalable SQL and NoSQL Data Stores
  – SIGMOD Record 2011
  – Cattell

• Dynamo: Amazon’s Highly Available Key-value Store
  – SOSP 2007
  – DeCandia et al

• MongoDB and Cassandra web sites
Synchronous Replication: Majority

- Majority voting technique guarantees consistency:
  - Xact must write a majority of copies to modify an object
  - Each copy has version number for object
  - Xact must read enough copies to be sure of seeing at least one most recent copy

- Example: 6 copies of data

Asynchronous Replication

- The modifying xact can commit before all copies have been changed
  - Users/apps must be aware of which copy they are reading, and that copies may be out-of-sync for short periods of time

- Two approaches for replication:
  - Primary Site
  - Peer-to-Peer (aka or update-anywhere)
  - Difference lies in how many copies are “updatable”

Cost of Synchronous Replication

- Before an update Xact can commit, it must obtain locks on all copies (assuming ROWA)
  - Sends lock requests to remote sites, and while waiting for the response, holds on to other locks!
  - If sites or links fail, Xact cannot commit until they are back up

So the alternative of asynchronous replication is becoming widely used

Primary Site Replication

- Exactly one copy of a relation partition is designated the primary copy.
  - Replicas at other sites cannot be directly updated

- How are changes to the primary copy propagated to the secondary copies?
  - One approach: log shipping
Peer-to-Peer Replication

- More than one of the copies of an object can be primary.
- Changes to a copy must be propagated to other copies.
- If two copies are updated in a conflicting manner, this must be resolved.
  - E.g., Last write wins? Combine updates somehow?

Strong vs. Eventual Consistency

- **Strong**: after update to an object, subsequent reads see that update.
- **Weak**: subsequent reads of an update may not reflect that update.
  - **Eventual**: if updates ceased, eventually the system would reflect all updates.

- **Eventual consistency** has some variation:
  - Read-your-own-writes, special case of session or causal consistency.
  - Monotonic reads.

> “Eventually Consistent - Building reliable distributed systems at a worldwide scale demands trade-offs between consistency and availability.”
> - Vogels, CTO Amazon.com

- BASE, not ACID!
  - BASE: Basically available, soft state, eventually consistent.

Werner Vogels on eventual consistency: http://www.allthingsdistributed.com/2008/12/eventually_conservent.html

Exercise 2: Replica Consistency

- Suppose have N replicas of some data object.
  - \( W = \# \text{replicas write to before xact commits} \)
  - \( R = \# \text{replicas read from} \)

- Strong consistency: overlap the \( W \) and \( R \) sets.
  - \( R + W > N \)
  - E.g., Read-one-write-all: \( R = 1, W = N \)

What is “NoSQL”?

*Web 2.0*

- Tons of user-generated content on the web.
  - E.g., social networking sites.
  - Web 1.0: few content creators, more static websites.

*Agile Development*

- Develop and change web applications iteratively.
  - Schema changes and non-uniformity.
  - Make changes while the application is live.

*Eventual Consistency (BASE not ACID)*

- Performance gains at the expense of consistency.
  - Grow incrementally by leveraging leased cloud resources like IaaS.
  - Automatically grow resources as needed.
Story of a “Successful” Web Startup

- Start with a relational DBMS running on single machine
  
  Web site gets popular!!
  Need to scale up...
  
  ... so manually partition/shard data across more nodes

- Logic in web application manages directing queries
  
  - Cross-shard filters and joins coded inside the app
  
  - App logic deals with data consistency

  As the number of machines increases, the chance that *something* fails increases

CAP Theorem

- Eric Brewer’s CAP theorem: distributed system can **only have two** of the following three properties:
  
  - Consistency
    
    Of replicated data
  
  - Availability
    
    For write requests
  
  - Tolerance to network *Partitions*

Tons of NoSQL systems

Summary: NoSQL Motivation

- Development of NoSQL systems motivated by difficulty **scaling up Web 2.0 applications**
  
  - Thousands to millions of users
  
  - Many [small] reads and writes (“small operations”)

- Typically **make sacrifices for performance**
  
  - E.g., no ACID xacts, eventual consistency
Achieving Scalable Performance

- Rule #1: Shared-nothing scalability
- Rule #4: High availability and automatic recovery essential
- Rule #5: On-line everything (system always “up”)
- Rule #6: Avoid multi-node operators

Rule #1: Shared-nothing scalability

- Goal: as application grows, need more servers added seamlessly
  - Don’t want manual management of scaling up
- Example techniques:
  - Consistent hashing (Dynamo and Cassandra)
  - Periodic re-balancing of partitions (MongoDB)

Rule #4: High Availability and Auto-Recovery

- Goal: updates always succeed!
  - Issue: conflicting writes on disjoint sets of replicas

Exercise 3

- Example: shopping cart
  - Add 2 items to cart, update goes to two replicas
  - Partition! Add 1 (different) item to each replica
  - Both carts are “version 2” 😊
- Dynamo: vector clocks

"...in the case of a timestamp tie, Cassandra follows two rules: first, deletes take precedence over inserts/updates. Second, if there are two updates, the one with the lexically larger value is selected.”

https://wiki.apache.org/cassandra/FAQ#clocktie
Replication/Availability Examples

- **MongoDB**: automatic failover for primary
  ![Diagram of MongoDB replication](https://docs.mongodb.org/master/MongoDB-replication-guide-master.pdf)

- **Cassandra/Dynamo**: peer-to-peer replication
  - tunable consistency, e.g., quorum or not

Rule #5: On-line Everything (Schema)

- **Recall**:
  - A **data model** is a collection of high-level data description constructs
  - A **schema** is a description of a particular collection of data, using a given data model

- Relational model has a rigid, **structured schema**
  - Attributes for relation pre-defined, shared by all tuples
  - Data and integrity constraints
  - Referential constraints

NoSQL: Non-Relational Data Models

- Agile development, live schema changes
  - No enforcement of structure
  - E.g., every “tuple” could have different attributes

- In essence, these data models are **key-based**
  - Key: some unique identifier to look up a corresponding “value”
  - What the value is can be complex

Key-Value Data Model

- **Example system**: Amazon’s Dynamo

- Key is some unique identifier, value can be anything, BLOB interpreted by app logic
  - E.g., id → shopping cart contents

- Query functionality
  - Get(key), put(key, value)
  - Only primary key index
  - No index lookups on non-keys (secondary indexes)
Document Data Model

- Example system: MongoDB
  - Stores collections of “documents” (e.g., JSON)
    - Relation: tuple :: Collection: document
    - Key ➔ Document
    - Document has key-value pairs, can be nested lists or scalars (and not defined in a global schema)
  - Query functionality
    - Primary key lookups
    - Secondary indexes on other attributes

Extensible Record (aka Column Family)

- Example system: BigTable, Cassandra
  - A bit more complex than document model
    - Relation: tuple :: ColumnFamily: Row
    - Key ➔ Set of columns (“wide-column store”)
    - Each column has key-value pairs
    - Different records can have different columns
  - Query functionality in “CQL”
    - Primary key lookups by row (with sorted columns)
    - Secondary indexes

MongoDB Example

Example: info about products, which have many parts

```
db.createCollection("parts")
db.createCollection("products")
```

```
// example part
{
  _id : ObjectID('AAAA'),
  partno : '123-aff-456',
  name : '#4 grommet',
  qty: 94,
  cost: 0.94,
  price: 3.99
  manufac_addr : [
    { street: '123 Sesame St',
      city: 'Anytown', cc: 'USA' },
    { street: '123 Avenue Q',
      city: 'New York', cc: 'USA' }]
}
```

What about a JOIN?

Modified from: http://blog.mongodb.org/post/87200945828/6-rules-of-thumb-for-mongodb-schema-design-part-1

Cassandra Examples

```
CREATE TABLE people (  
  user_id text PRIMARY KEY,  
  name text,  
  addresses list  
);
```

```
CREATE TABLE comments (  
  article_id uuid,  
  posted_at timestamp,  
  author text,  
  content text,  
  PRIMARY KEY (article_id,posted_at)  
);
```

```
<table>
<thead>
<tr>
<th>alicious</th>
<th>addresses</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>West</td>
<td>Alice</td>
</tr>
<tr>
<td></td>
<td>North</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Drinkward</td>
<td></td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>bobtastic</th>
<th>addresses</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 Lonely Ave</td>
<td>Bob</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>2015-12-03</th>
<th>2015-11-11</th>
</tr>
</thead>
<tbody>
<tr>
<td>author Bob</td>
<td>author Alice</td>
</tr>
<tr>
<td>content Bravo</td>
<td>content Awful</td>
</tr>
</tbody>
</table>
```

```
```

Roughly based on: http://www.datastax.com/dev/blog/thrift-to-cql3
Rule #6: Avoid Multi-Node Queries

- No ACID transactions across primary keys

- No joins! **Denormalization** helps

- Systems offer different levels of “protection”
  - Key-value stores: `get (key)` method requires key
  - MongoDB: Table scans **discouraged**
  - Cassandra: Table scans **prohibited**