CS 133: Databases

Fall 2019
Lec 25 – 12/10
NoSQL

Prof. Beth Trushkowsky

Final Exam Logistics

- Final exam take-home
  - Available: in-class on Thursday
  - Due: Wednesday December 18th, 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

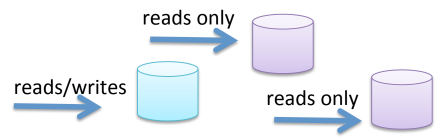
- *Ten Rules for Scalable Performance in "Simple Operation" Datastores*
  - Communications of the ACM 2011
  - Stonebraker and Cattell
- *Scalable SQL and NoSQL Data Stores*
  - SIGMOD Record 2011
  - Cattell
- *Dynamo: Amazon's Highly Available Key-value Store*
  - SOSP 2007
  - DeCandia et al
- MongoDB and Cassandra web sites

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”**

Primary Site Replication

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

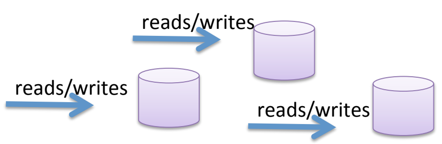


If reads happen at secondary copies, then possible that a xact is not be able to ***read its own writes***

- 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_consistent.html

Exercise 2: Replica Consistency

- Suppose have N replicas of some data object
  - W = # replicas write to before xact commits
  - R = # 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"?

A *movement* around **non-relational** data stores

Here's a challenge: can you describe NoSQL without using *any* buzz words?!

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)

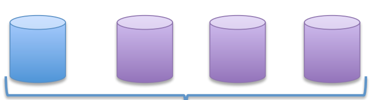
Performance gains at the expense of consistency

- 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…

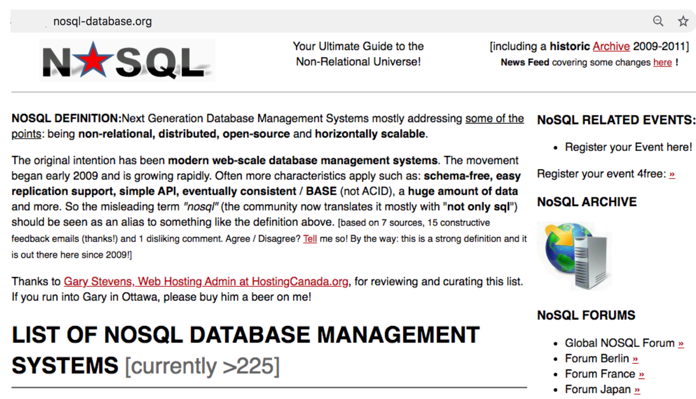


… so manually partition/shard data across more nodes

- 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

Tons of NoSQL systems



nosql-database.org

NOSQL

Your Ultimate Guide to the Non-Relational Universe!

[including a **historic** Archive 2009-2011]
News Feed covering some changes here !

NOSQL DEFINITION: Next Generation Database Management Systems mostly addressing some of the points: being **non-relational, distributed, open-source** and **horizontally scalable**.

The original intention has been **modern web-scale database management systems**. The movement began early 2009 and is growing rapidly. Often more characteristics apply such as: **schema-free, easy replication support, simple API, eventually consistent / BASE** (not ACID), a **huge amount of data** and more. So the misleading term *"nosql"* (the community now translates it mostly with "**not only sql**") should be seen as an alias to something like the definition above. [based on 7 sources, 15 constructive feedback emails (thanks!) and 1 disliking comment. Agree / Disagree? Tell me so! By the way: this is a strong definition and it is out there here since 2009!]

Thanks to Gary Stevens, Web Hosting Admin at HostingCanada.org, for reviewing and curating this list. If you run into Gary in Ottawa, please buy him a beer on me!

LIST OF NOSQL DATABASE MANAGEMENT SYSTEMS [currently >225]

NoSQL RELATED EVENTS:

- Register your Event here!

Register your event 4free: »

NoSQL ARCHIVE

NoSQL FORUMS

- Global NOSQL Forum »
- Forum Berlin »
- Forum France »
- Forum Japan »

CAP Theorem

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

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

Stonebraker and Cattell

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)

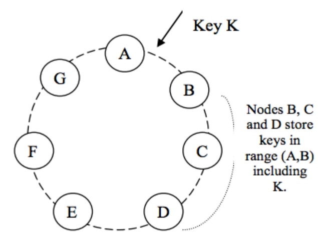


Fig 2 in Dynamo

Rule #4: High Availability and Auto-Recovery

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

Exercise 3

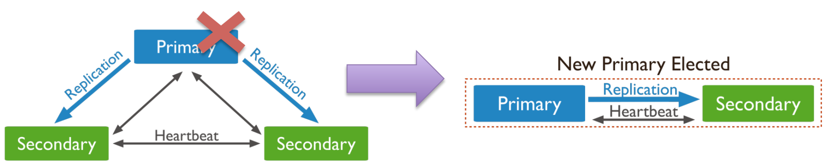
- Example: shopping cart
  - Add 2 items to cart, update goes to two replicas
  - Partition! Add 1 (different) item to each replica
  - Both carts are "version 2" ☹
- Dynamo: **vector clocks**
- Or latest timestamp wins?

> "…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



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

Pics from https://docs.mongodb.org/master/MongoDB-replication-guide-master.pdf

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

What if you want more flexibility?

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 typically plays a role in data partitioning scheme

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

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' }]
}
```

```
// example product
{
  name : 'smoke shifter',
  manufacturer : 'Acme Corp',
  catalog_number: 1234,
  parts : [
    ObjectID('AAAA'),
    ObjectID('F17C'),
    ObjectID('D2AA')]
}
```

What about a JOIN?

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

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

“Row” called “partition” now

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***

Cassandra Examples

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

Becomes the partition key

alicious	addresses	name
	West	Alice
	North	
	Drinkward	
bobtastic	**addresses**	**name**
	1 Lonely Ave	Bob

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

First argument is partition key, others are “clustering” columns

a4i89	2015-12-03		…	2015-11-11	
	author	Bob		author	Alice
	content	Bravo!		content	Awful ☹

Roughly based on: http://www.datastax.com/dev/blog/thrift-to-cql3