Introduction to Database Systems
CSE 414

Lecture 29: NoSQL

Announcements

- HW8 due Friday, 11 pm
  - **No late assignments. At all.**
- Final exam:
  - Mon. 6/10, 2:30-4:20, this room
  - Comprehensive
  - Same rules as before: open textbook + 1 sheet of handwritten notes (+ midterm sheet), nothing else
- Review:
  - In sections tomorrow
  - Extra Q&A Sunday, 6/9, 2 pm, Loew 101

Where We Are

- Well… we are nearly done
- No more web quizzes
- Only hw8 left

- Friday: course review and final exam topics

- Today: NoSQL!

References

- **Scalable SQL and NoSQL Data Stores**, Rick Cattell, SIGMOD Record, December 2010 (Vol. 39, No. 4)
- **Bigtable: A Distributed Storage System for Structured Data**. Fay Chang, Jeffrey Dean, et. al. OSDI 2006
- Online documentation: Amazon SimpleDB, Google App Engine Datastore, etc.

NoSQL Motivation

- Originally motivated by Web 2.0 applications
- Goal is to scale simple OLTP-style workloads to thousands or millions of users
- Users are doing both updates and reads

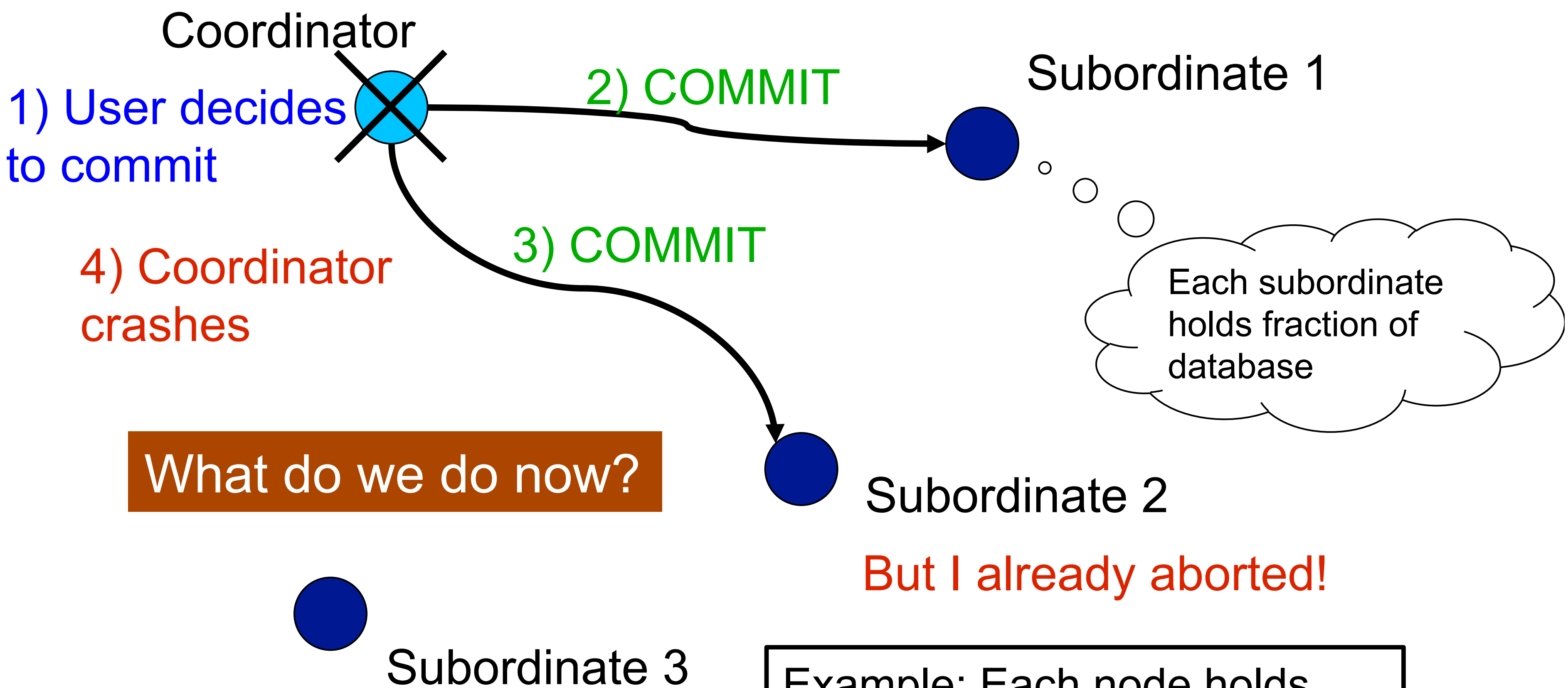
What is the Problem?

- Scaling a relational DBMS is hard
- We saw how to scale queries with parallel DBMSs
- Much more difficult to scale ***transactions***
- ***Because need to ensure ACID properties***
  - Hard to do beyond a single machine

Scaling Transactions

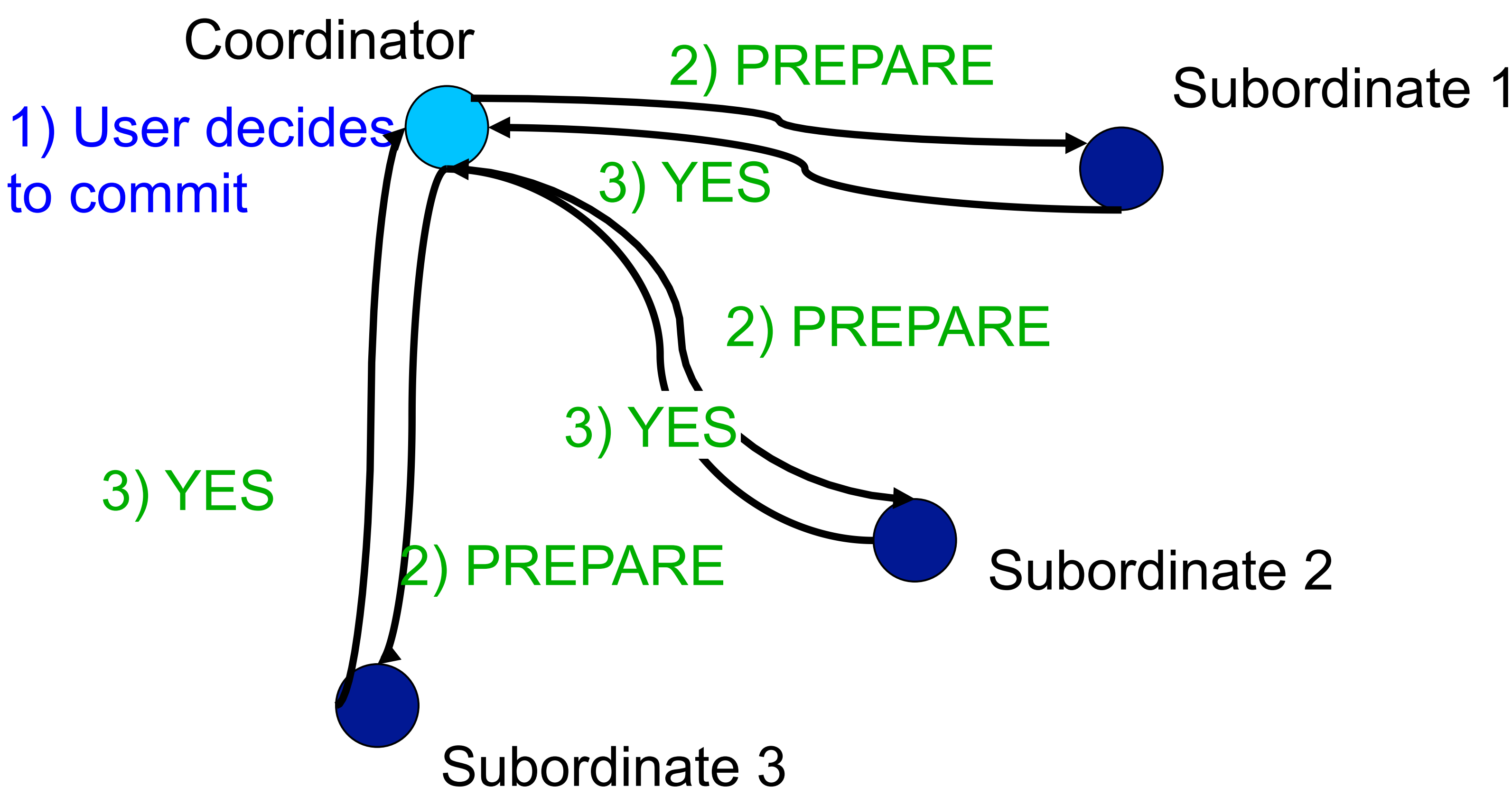
- Need to partition the db across machines
- If a transaction touches one machine
  - Life is good
- If a transaction touches multiple machines
  - ACID becomes extremely expensive!
  - Need two-phase commit

Two-Phase Commit: Motivation

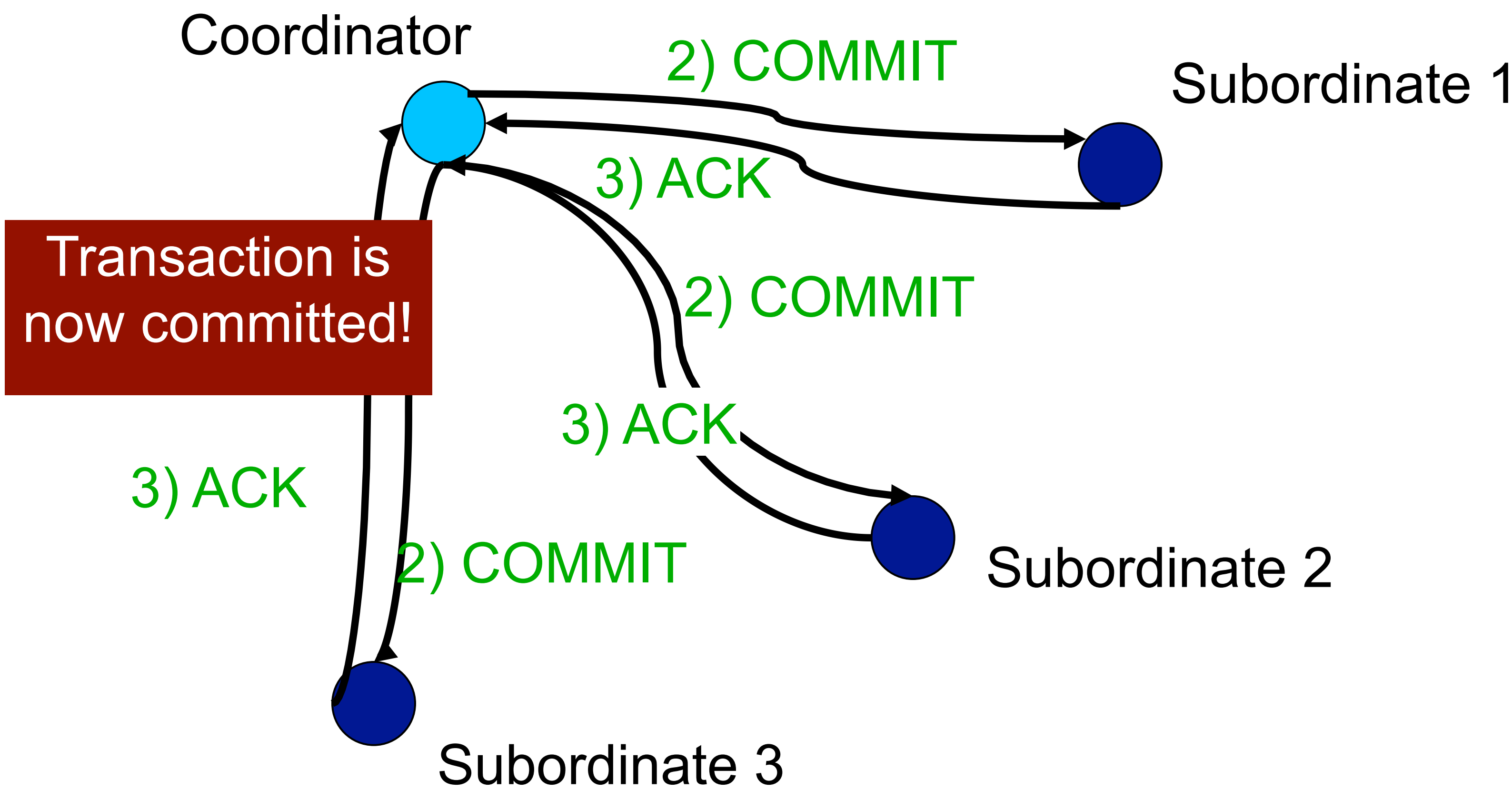


Example: Each node holds some subset of bank accounts
Transaction transfers money

2PC: Phase 1 Illustrated

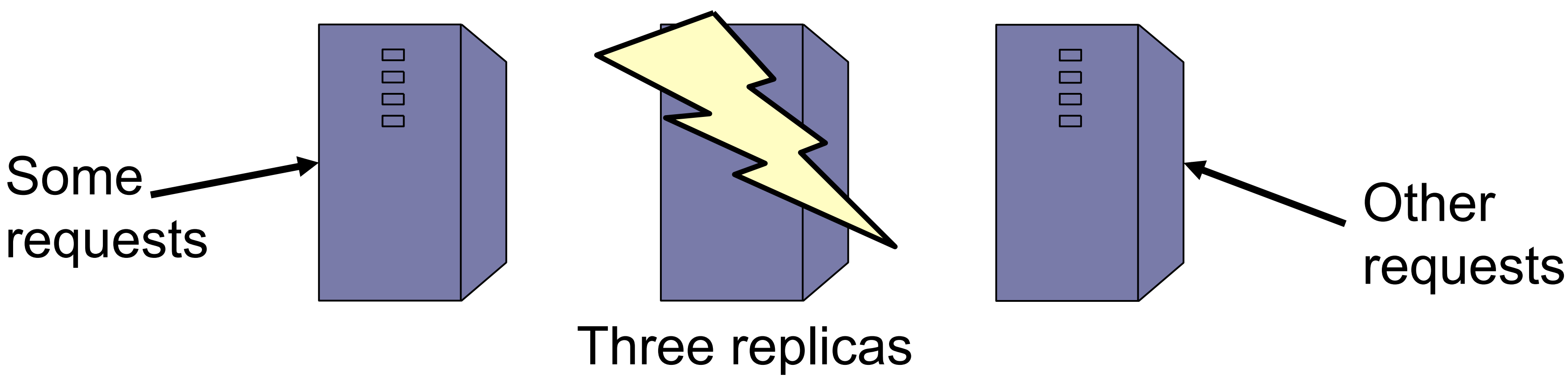


2PC: Phase 2 Illustrated



Scale Through Replication?

- Create multiple copies of each database partition
- Spread queries across these replicas
- Can increase throughput and lower latency
- Easy for reads but writes, once again, become expensive!



NoSQL Key Feature Decisions

- Want a data management system that is
  - Elastic and highly scalable
  - Flexible (different records have different schemas)
- To achieve above goals, willing to give up
  - Complex queries: e.g., give up on joins
  - Multi-object transactions
  - ACID guarantees: e.g., eventual consistency is OK
  - *Not all NoSQL systems give up all these properties*

Cattell, SIGMOD Record 2010

“Not Only SQL” or “Not Relational”

Six key features:

1. Scale horizontally “simple operations”
   – key lookups, reads and writes of one record or a small number of records, simple selections
2. Replicate/distribute data over many servers
3. Simple call level interface (contrast w/ SQL)
4. Weaker concurrency model than ACID
5. Efficient use of distributed indexes and RAM
6. Flexible schema

Cattell, SIGMOD Record 2010

Terminology

- Sharding = horizontal partitioning by some key, and storing records on different servers in order to improve performance

- Horizontal scalability = distribute both data *and* load over many servers

Scale-out

- Vertical scaling = when a dbms uses multiple cores and/or CPUs

Scale-up

ACID Vs BASE

- ACID = Atomicity, Consistency, Isolation, and Durability
- BASE = Basically Available, Soft state, Eventually consistent

Data Models

- Tuple = row in a relational database
- Document = nested values, extensible records (think XML, JSON, attribute-value pairs)
- Extensible record = families of attributes have a schema, but new attributes may be added
- Object = like in a programming language, but without methods

Cattell, SIGMOD Record 2010

Different Types of NoSQL

Taxonomy based on data models:

- Key-value stores
  - e.g., Project Voldemort, Memcached
- Document stores
  - e.g., SimpleDB, CouchDB, MongoDB
- Extensible Record Stores
  - e.g., HBase, Cassandra, PNUTS

Key-Value Stores Features

- **Data model**: (key,value) pairs
  - A single key-value index for all the data
- **Operations**
  - Insert, delete, and lookup operations on keys
- **Distribution / Partitioning**
  - Distribute keys across different nodes
- **Other features**
  - Versioning
  - Sorting

Key-Value Stores Internals

- Data remains in main memory
- One type of impl.: distributed hash table
- Most systems also offer a persistence option
- Others use replication to provide fault-tolerance
  - Asynchronous or synchronous replication
  - Tunable consistency: read/write one replica or majority
- Some offer ACID transactions others do not
- Multiversion concurrency control or locking

Cattell, SIGMOD Record 2010

Different Types of NoSQL

Taxonomy based on data models:

- Key-value stores
  - e.g., Project Voldemort, Memcached
- Document stores
  - e.g., SimpleDB, CouchDB, MongoDB
- Extensible Record Stores
  - e.g., HBase, Cassandra, PNUTS

Amazon SimpleDB

A Document Store

- **Partitioning**
  - Data partitioned into domains: queries run within a domain
  - Domains seem to be unit of replication. Limit 10GB
  - Can use domains to manually create parallelism
- **Data Model / Schema**
  - No fixed schema
  - Objects are defined with attribute-value pairs

Amazon SimpleDB (2/3)

- **Indexing**
  - Automatically indexes all attributes
- **Support for writing**
  - PUT and DELETE items in a domain
- **Support for querying**
  - GET by key
  - Selection + sort
  - A simple form of aggregation: count
  - Query is limited to 5s and 1MB output (but can continue)

```
select output_list
from domain_name
[where expression]
[sort_instructions]
[limit limit]
```

Amazon SimpleDB (3/3)

- **Availability and consistency**
  - “Fully indexed data is stored redundantly across multiple servers and data centers”
  - “Takes time for the update to propagate to all storage locations. The data will eventually be consistent, but an immediate read might not show the change”
  - Today, can choose between consistent or eventually consistent read

Cattell, SIGMOD Record 2010

Different Types of NoSQL

Taxonomy based on data models:

- Key-value stores
  - e.g., Project Voldemort, Memcached
- Document stores
  - e.g., SimpleDB, CouchDB, MongoDB
- Extensible Record Stores
  - e.g., HBase, Cassandra, PNUTS

Extensible Record Stores

- Based on Google's BigTable
- Data model is rows and columns
- Scalability by splitting rows and columns over nodes
  - Rows partitioned through sharding on primary key
  - Columns of a table are distributed over multiple nodes by using "column groups"
- HBase is an open source implementation of BigTable

What is Bigtable?

- Distributed storage system
- Designed to
  - Hold structured data
  - Scale to thousands of servers
  - Store up to several hundred TB (maybe even PB)
  - Perform backend bulk processing
  - Perform real-time data serving
- To scale, Bigtable has a limited set of features

Chang, OSDI 2006

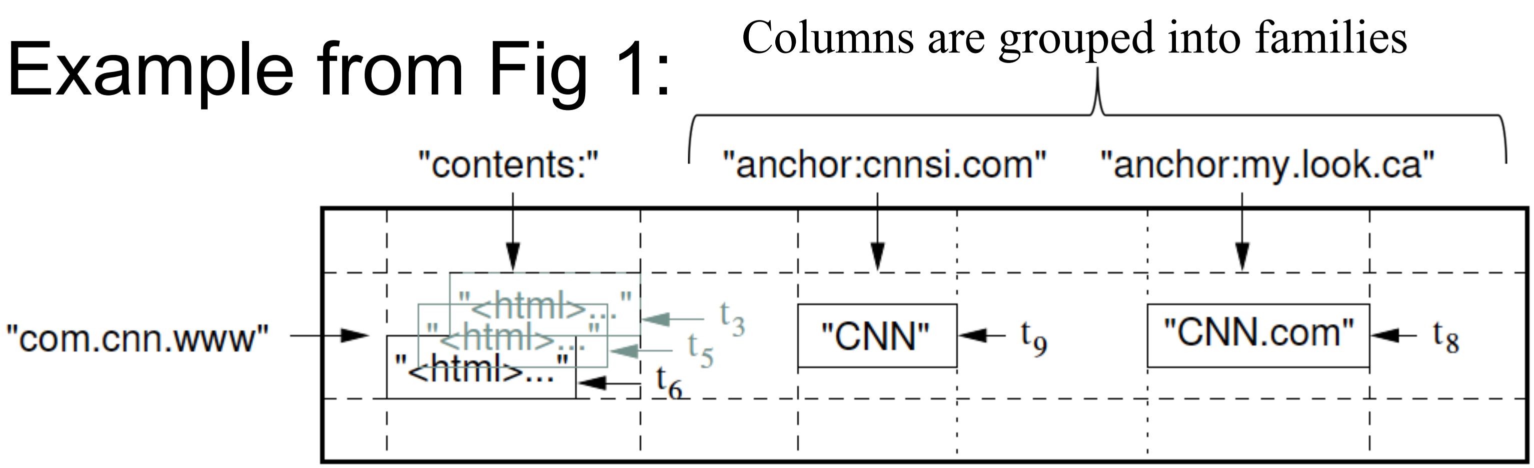
Bigtable Data Model

- Sparse, multidimensional sorted map

  `(row:string, column:string, time:int64)` ➔ `string`

  Notice how everything but time is a string

- Example from Fig 1:



Chang, OSDI 2006

BigTable Key Features

- Read/writes of data under single row key is atomic
  - Only single-row transactions!
- Data is stored in lexicographical order
  - Improves data access locality
- Column families are unit of access control
- Data is versioned (old versions garbage collected)
  - Ex: most recent three crawls of each page, with times

Chang, OSDI 2006

BigTable API

- Data definition
  - Creating/deleting tables or column families
  - Changing access control rights
- Data manipulation
  - Writing or deleting values
  - Supports single-row transactions
  - Looking up values from individual rows
  - Iterating over subset of data in the table
    - Can select on rows, columns, and timestamps

Cattell, SIGMOD Record 2010

Megastore

- BigTable is implemented, used within Google
- Megastore is a layer on top of BigTable
  - Transactions that span nodes
  - A database schema defined in a SQL-like language
  - Hierarchical paths that allow some limited joins
- Megastore is made available through the Google App Engine Datastore

Google App Engine

- "Run your web applications on Google's infrastructure"

- Limitation: app must be written in Python or Java
- Key features (examples for Java)
  - A complete development stack that uses familiar technologies to build and host web applications
  - Includes: Java 6 JVM, a Java Servlets interface, and support for standard interfaces to the App Engine scalable datastore and services, such as JDO, JPA, JavaMail, and Jcache
  - JVM runs in a secured "sandbox" environment to isolate your application for service and security (some ops not allowed)

Google App Engine Datastore (1/3)

- “Distributed data storage service that features a query engine and transactions”
- **Partitioning**
  - Data partitioned into “entity groups”
  - Entities of the same group are stored together for efficient execution of transactions
- **Data Model / Schema**
  - Each entity has a key and properties that can be either
    - Named values of one of several supported data types (includes list)
    - References to other entities
  - Flexible schema: different entities can have different properties

Google App Engine Datastore (2/3)

- **Indexing**
  - Applications define indexes: must have one index per query type
- **Support for writing**
  - PUT and DELETE entities (for Java, hidden behind JDO)
- **Support for querying**
  - GET an entity using its key
  - Execute a query: selection + sort
  - Language bindings: invoke methods or write SQL-like queries
  - Lazy query evaluation: query executes when user accesses results

Google App Engine Datastore (3/3)

- **Availability and consistency**
  - Every datastore write operation (put/delete) is atomic
    - Outside of transactions, get READ_COMMITTED isolation
  - Support transactions (many ops on many objects)
    - Single-group transactions
    - Cross-group transactions with up to 5 groups
    - Transactions use snapshot isolation
    - Transactions use optimistic concurrency control

Cattell, SIGMOD Record 2010

Different Types of NoSQL

Taxonomy based on data models:

- Key-value stores
  - e.g., Project Voldemort, Memcached
- Document stores
  - e.g., SimpleDB, CouchDB, MongoDB
- Extensible Record Stores
  - e.g., HBase, Cassandra, PNUTS