Introduction to Data Management
CSE 344

Lecture 26:
Data Integration
and
NoSQL

Where We Are

- Well… we are nearly done
- Only hw8 left (due tomorrow, 1 late day allowed)
- Let us know asap if you have any question about assignment grades.

- **Today: Parallel DB wrapup, data integration, NoSQL**
  – Data integration, NoSQL: understand “why”, not all details of “how”

- **Friday: Final review**
  – Everything up to Friday’s lecture is in the exam

Remaining Review Sessions

- Today's class (3/12): Parallel DB/MR
- Tomorrow's sections (3/13): Transaction
- Friday's class (3/14): Entire material

- Saturday (3/15) 2-4 pm, CSE 303: Additional office hour (sudeepa) for any question
  - also Thursday (3/13), 4:30-5:30, CSE 344

- Even if you missed the review sessions, go through the practice problems in the notes

Final Exam

- Tuesday, March 18, 2:30-4:30 pm, in class
- Content: **Everything**
  - But focus on Lectures 16 through 27 more
- Open books and open notes
  - You can bring **any notes** that you want including slides (check for updates), assignments, old exams, stuff from the web, …
  - BUT NO laptops, phones, or other devices

How To Study

- Go over the lecture notes
- Read the book
- Go over the homeworks (your graded solution + ours)
- Practice (within stipulated time)
  - Practice web quiz (entire question explanation visible now)
  - Finals from past 344
  - Look at both midterms and finals from 444 past years: be careful because several questions do not apply to us!
  - Questions in the book
  - Some old finals have 100 points, some 200 points, budget your time
- **Ask course staff questions on discussion board**
- The goal of the final is to help you learn

Data Integration

Today: Data Integration

- Goal:
  - Data is available in multiple distinct databases
  - Want to ask queries across all these databases
- When is data integration needed?
  - Two companies merge
    - United/Continental, BOA buying FleetBoston
  - Want to get legacy databases to talk to each other
    - Univ registration vs. bursar db
  - Want to analyze data produced by different sources
    - Company vs. Scientific db
  - Want to combine data from different websites
    - yelp, tripadvisor, Hotels.com, kayak

Data Integration Challenges (1/2)

- Each database could be in a different type of DBMS (different data model, query language, etc.)
  - Relational, semi-structured, NoSQL
- Schema heterogeneity
  - S1: Employee(ID, name, address, position, salary)
  - S2: Worker(EID, name, address) Position(PID,salary,from,until)
- Data type heterogeneity
  - Employee ID could be a string or an integer
- Value heterogeneity
  - The “cashier” position could be called “cashier” or “associate”

Data Integration Challenges (2/2)

- Semantic heterogeneity
  - Most difficult to manage
  - E.g., salary is hourly salary before tax
  - Or salary is net, weekly salary with lunch allowance
- Data integration is a very, very, very difficult task!

Data Integration Approaches

- Federated databases
  - Each source remains independently administered
  - One source can call on others to supply info
- Centralized warehouse
  - Data from source is **extracted-transformed-loaded** into a single, centralized database (same global schema)
  - Data is refreshed periodically (so data is not 100% up-to-date)
  - Very popular for data analytics, but hard to add new sources
- Mediator
  - Virtual database on top of others
  - Takes query as input and rewrites it in terms of queries over the other databases, then synthesizes the answer (extraction/transformation/loading are done at query time)
  - Data is always up-to-date.

Creating a Data Warehouse

- Extract data from distributed operational databases
  - Can do this by running a query over the data source
- Clean to minimize errors and fill in missing information
- Transform to reconcile semantic (and other) mismatches
  - Performed by defining views over the data sources
- Load to materialize the above defined views
  - Build indexes and additional materialized views
- Refresh to propagate updates to warehouse periodically
  - Update the warehouse *incrementally*

Executing Queries over Mediator

- Wrappers more complex than with warehouses
  - Need to execute all sorts of queries, not just extract
  - One approach is to define *templates*
  - A template is a parameterized query
    - select * from EmployeeMed where position='$p'
- Query optimization at the mediator is a challenge
  - Wrapped data sources can be seen as views
  - **How can I answer the given query using these views?**
- Data sources can exhibit bad and variable performance
  - May want a more dynamic query plan: process data as it arrives

Dirty Data

- Another challenge with data integration
  - Often hard to decide if two records represent the same entity or not
  - E.g. Univ. of Washington, UW, UWash, ….
  - E.g.,
    - John Doe from 1234 56th ave NE, Seattle
    - Vs. J. R. Doe from 1234 56th ave NE, Seattle
    - Vs John Doe from 789 108th St., Bellevue

- Even without data integration, data often dirty
  - Missing values, duplicates, odd characters, etc.
  - **Data Cleaning** is another independent research area

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

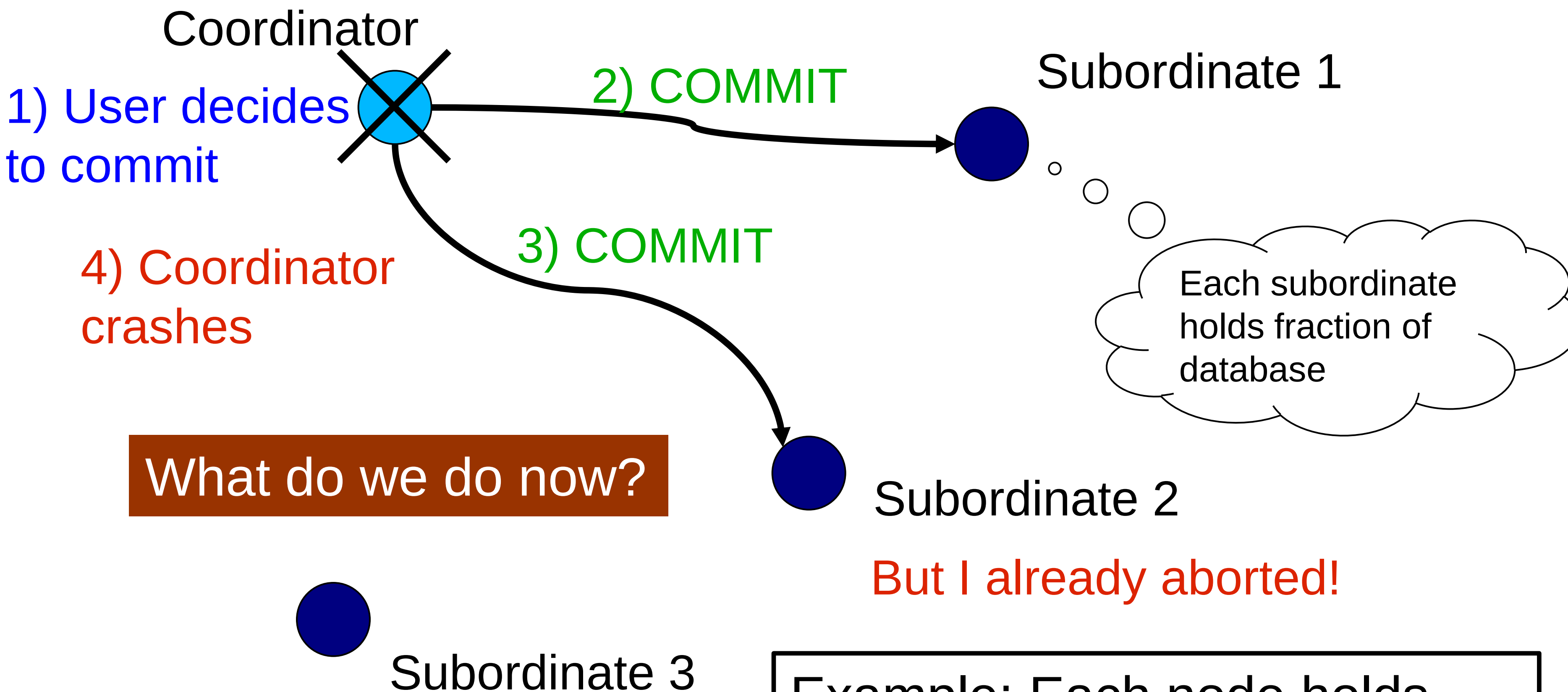
- What is the problem with Rel. DB/SQL?
  - Scaling a relational DBMS is hard
  - We saw how to scale queries with parallel DBMSs
  - Much more difficult to scale ***transactions***
  - ***Because need to ensure ACID properties***
    - Hard to do beyond a single machine

- NoSQL
  - 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

Scaling Transactions

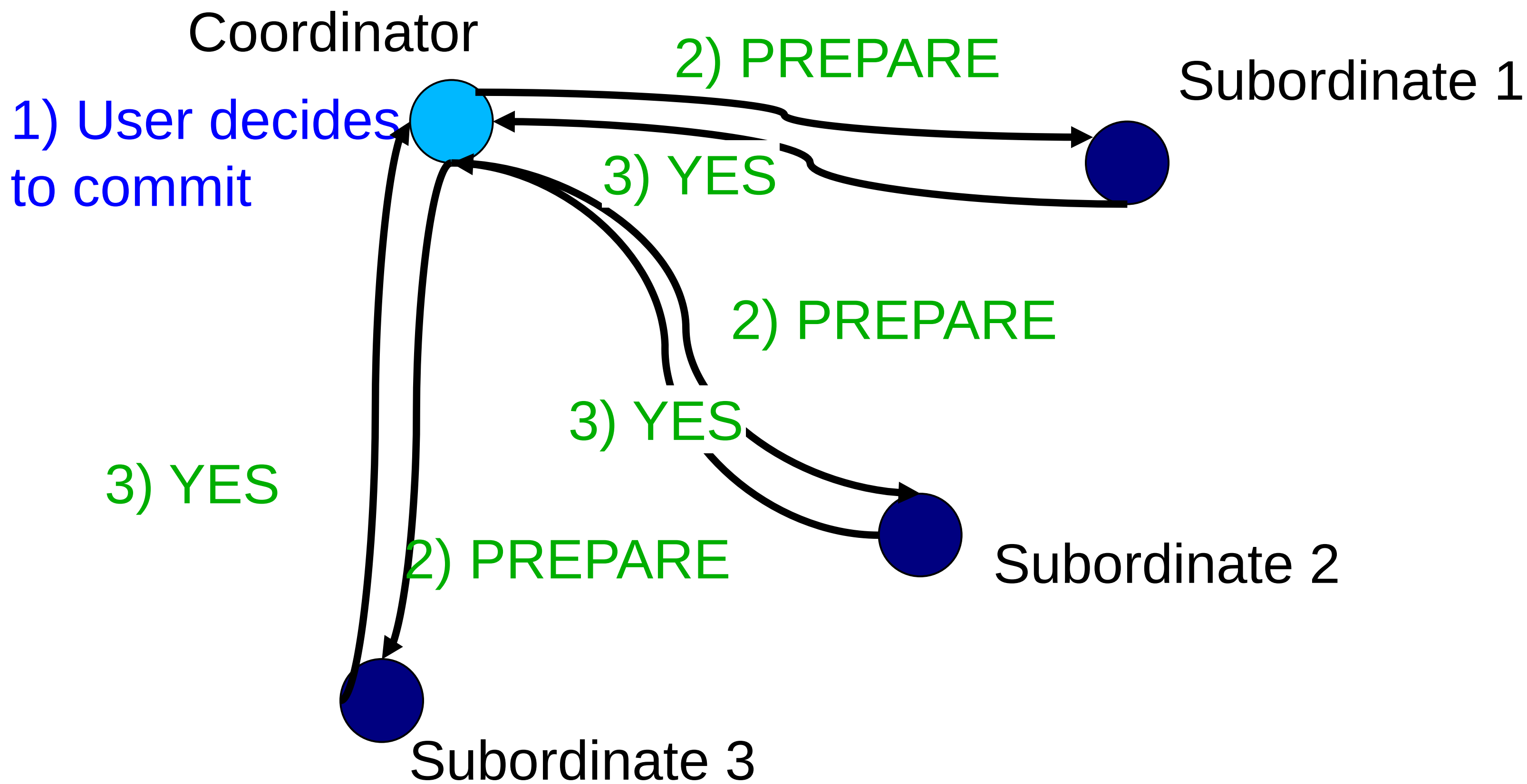
- Need to partition the db across machines
- If a transaction touches one machine
  - Life is good
- If a transaction touches multiple machines
  - ACID becomes extremely expensive!
  - Need two-phase commit (in details in cse 444)

Two-Phase Commit: Motivation

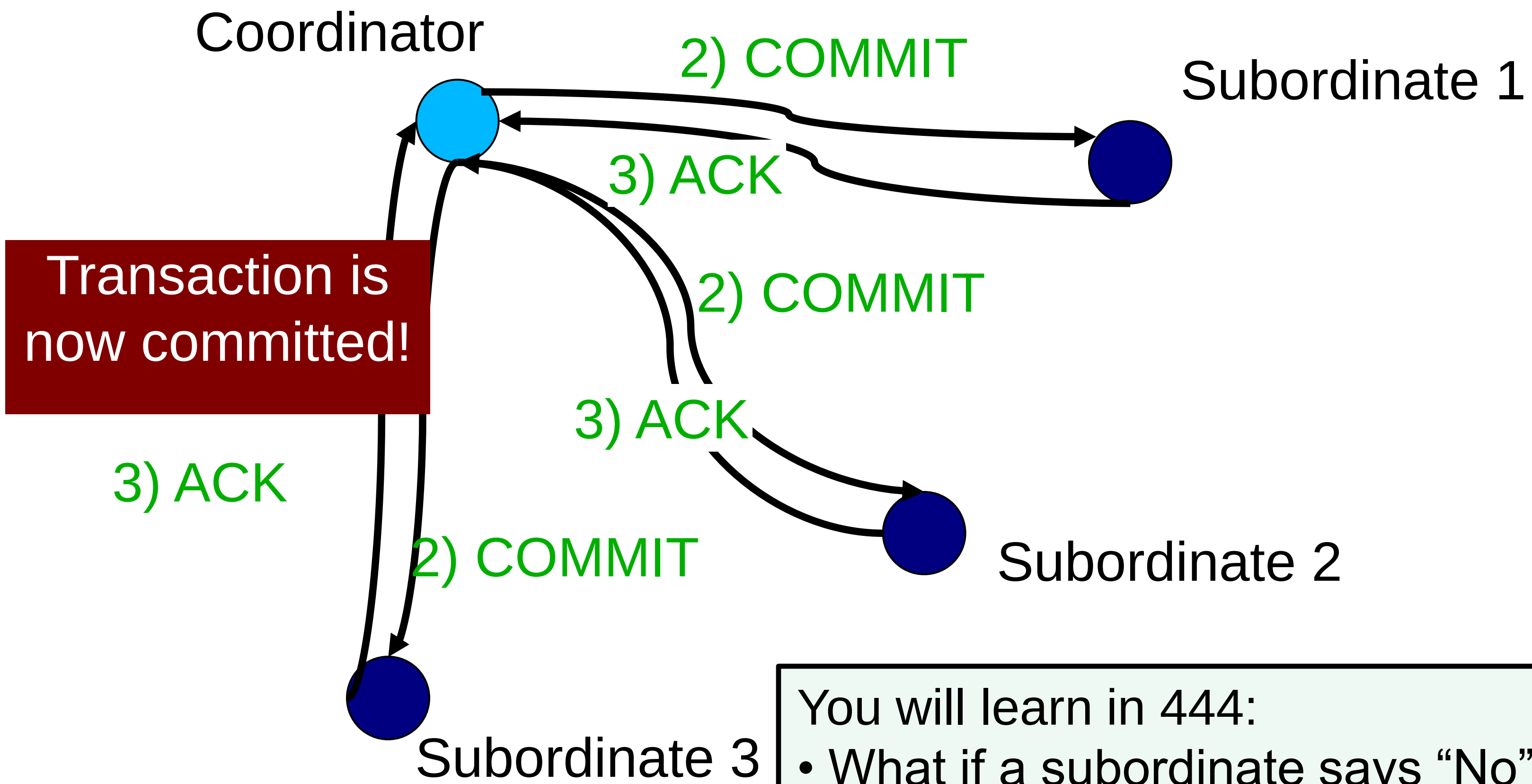


Example: Each node holds some subset of bank accounts
Transaction transfers money

2PC: Phase 1 Illustrated



2PC: Phase 2 Illustrated

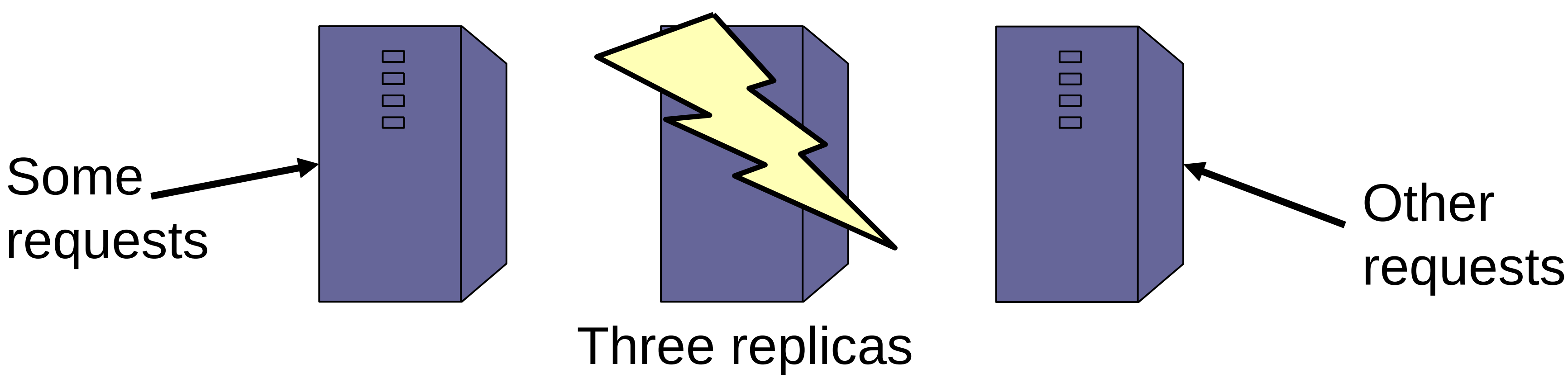


You will learn in 444:

- What if a subordinate says “No”?
- What is written to log?
- What if coordinator/subordinate crashes?

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

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 (LinkedIn), Memcached
- Document stores
  - e.g., SimpleDB (Amazon), CouchDB, MongoDB
- Extensible Record Stores
  - e.g., BIGTABLE(Google), HBase, Cassandra, PNUTS

Additional slides on NoSQL (optional)

Cattell, SIGMOD Record 2010

Different Types of NoSQL

Taxonomy based on data models:

- **Key-value stores (next)**
  - 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

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

Cattell, SIGMOD Record 2010

Different Types of NoSQL

Taxonomy based on data models:

- Key-value stores
  - e.g., Project Voldemort, Memcached
- **Document stores (next)**
  - 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 (next)**
  - 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

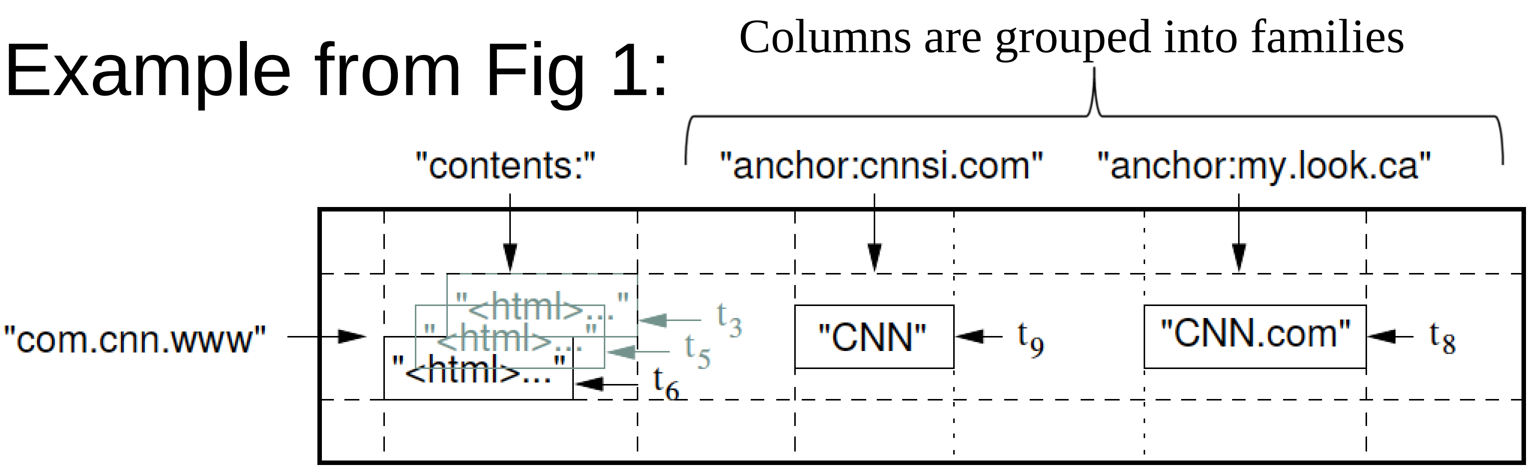
Bigtable Data Model

- 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