CIS 560 – Database System Concepts

Lecture 34

December 4, 2013

Credits for slides: Lakshman, Malik, Evans, Ellis Widom, Gabor

Where we are

- Last:
  - Key-Value Stores
    - Dynamo: Amazon's Highly Available Key-value Store (2007)
  - Wide-Column Stores or Column Families
    - Bigtable: A Distributed Storage System for Structured Data
- Today:
  - Cassandra
    - Cassandra - A Decentralized Structured Storage System
  - Document stores
    - MongoDB, CouchDB

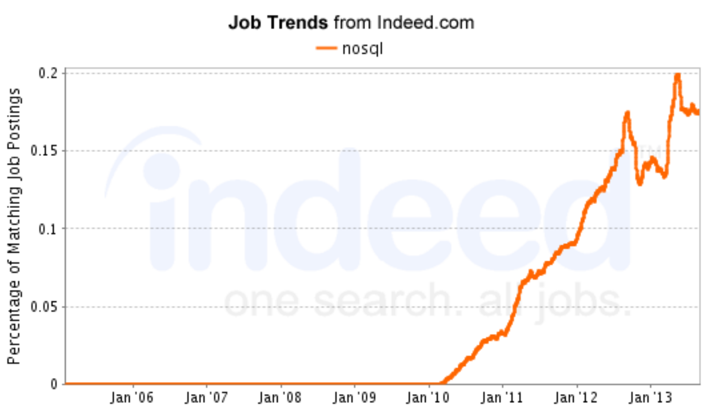
Cassandra Job Trends

(Indeed.com searches millions of jobs from thousands of job sites.)

MongoDB Job Trends

Big Data Trends

NoSQL Job Trends



Relational Database Trends

Influential Papers

- BigTable
  - Strong consistency (master-slave architecture)
  - Distributed, sparse map data model
- Dynamo
  - O(1) distributed hash table (DHT)
  - BASE (aka eventual consistency)
  - Client tunable consistency/availability

Why Cassandra?

- Lots of data
  - E.g. application: Inbox search - copies of messages, inverted indexes of messages, per user data.
- Many incoming requests resulting in a lot of random reads and random writes.
- No existing production ready solutions in the market meet these requirements.

Design Goals

- High availability
- Eventual consistency
  - trade-off strong consistency in favor of high availability
- Incremental scalability (linearly scalable)
- Flexible partitioning, replica placement
- No single point of failure
- “Knobs” to tune tradeoffs between consistency, durability and latency
- Low total cost of ownership
- Minimal administration

No single point of failure

Some Cassandra Users

Industries and use cases

- Financial
- Social Media
- Advertising
- Entertainment
- Energy
- E-tail
- Health care
- Government

- Time series data
- Messaging
- Ad tracking
- Data mining
- User activity streams
- User sessions
- Anything requiring: **Scalability + high availability**

Netflix

Application/Use Case

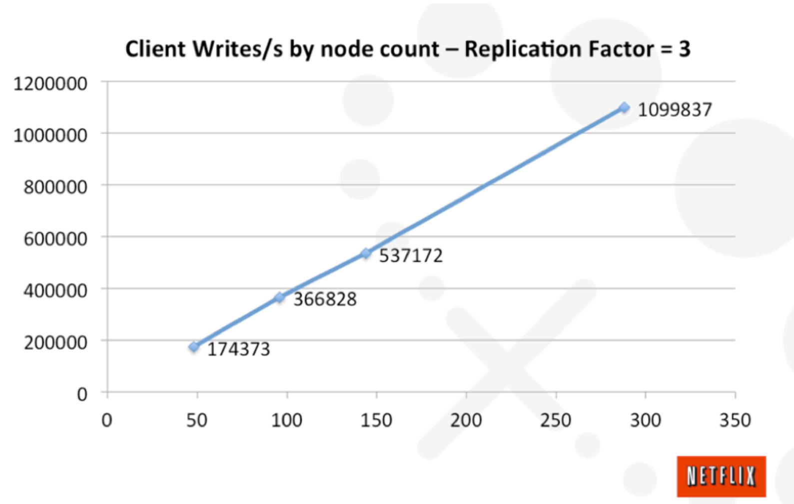
- Manage subscriber interactions with downloaded movies
- Need to handle distributed databases all over the world (40 countries)
- Need better TCO than Oracle

Why Cassandra?

- Easy scale and multi-data center support for geographical data distribution
- Data model perfect fit for customer interaction data
- Much better TCO than Oracle or SimpleDB

"I can create a Cassandra cluster in any region of the world in 10 minutes. When marketing guys decide we want to move into a certain part of the world, we're ready."

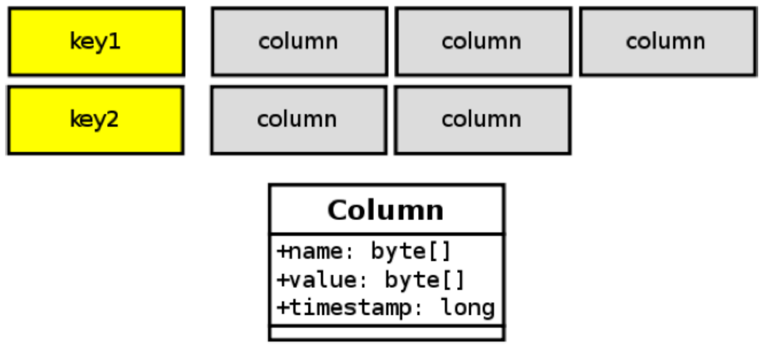
Scale-up linearity



Data Model - Overview

- Distributed, multi-dimensional map index by a key; the value is a highly structured object.
- ColumnFamily
  - Associates records of a similar kind
  - Record-level atomicity
  - Indexed
  - Simple/Super
- Column
  - Basic unit of storage
  - Sorted by time or name

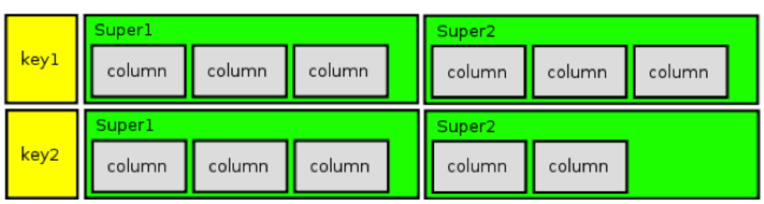
Column Families



Column sorted by time or name

Super-column families

Column family within a column family



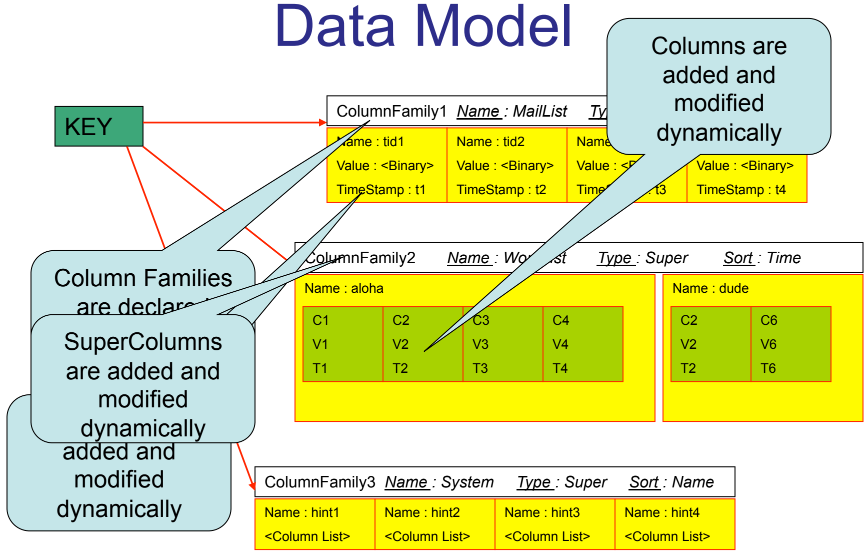
Inbox Search

- Maintain per user index of all messages that have been exchanged between the sender and the recipients.
- Search:
  - Find messages containing given term
  - Interactions: given the name of a person, return all messages that the user might have ever sent to or received from that person
- Schema consists of two super-columns

Inbox Search – Super-columns

- Term search
  - Key: User ID
  - Super-column: words that make up the message
  - Column: Individual message identifiers of the messages that contain the term become the columns within the super-column
- Interactions
  - Key: User ID
  - Super-column: recipients IDs
  - Column: individual message identifiers

Data Model



Performance Benchmark

- Loading of data - limited by network bandwidth.
- Read performance for Inbox Search in production:

	Search Interactions	Term Search
Min	7.69 ms	7.78 ms
Median	15.69 ms	18.27 ms
Average	26.13 ms	44.41 ms

API

- *insert(table,key,rowMutation)*
- *get(table,key,columnName)*
- *delete(table,key,columnName)*

Querying

- By column
- By column for multiple keys
- Slice by names, or ranges of names
  - returning columns
  - returning super-columns
- Slice for multiple keys
- Range of keys
- Slice on a key range RSN

MySQL Comparison

- MySQL > 50 GB Data
  Writes Average : ~300 ms
  Reads Average : ~350 ms
- Cassandra > 50 GB Data
  Writes Average : 0.12 ms
  Reads Average : 15 ms

Lessons

- Add fancy features only when absolutely required.
- Many types of failures are possible.
- Big systems need proper systems-level monitoring.
- Value simple designs

Document-based key-value stores

Document

Key ->

```
{
    "day": [ 2010, 01, 23 ],
    "products": {
        "apple": {
            "price": 10
            "quantity": 6
        },
        "kiwi": {
            "price": 20
            "quantity": 2
        }
    },
    "checkout": 100
}
```

	Couchdb	Mongodb
Data Model	Document-Oriented (JSON)	Document-Oriented (BSON)
Interface	HTTP/REST	Custom protocol over TCP/IP
Object Storage	Database contains Documents	Database contains Collections Collections contains Documents
Query Method	Map/Reduce (javascript + others) creating Views + Range queries	Map/Reduce (javascript) creating Collections + Object-Based query language
Replication	Master-Master with custom conflict resolution functions	Master-Slave
Concurrency	MVCC (Multi Version Concurrency Control)	Update in-place
Written In	Erlang	C++

JSON

JavaScript Object Notation (JSON)

- Standard for "serializing" data objects, usually in files
- Human-readable, useful for data interchange
- Also useful for representing & storing semi-structured data


```
{ "Books":
  [
    { "ISBN":"ISBN-0-13-713526-2",
      "Price":85,
      "Edition":3,
      "Title":"A First Course in Database Systems",
      "Authors":[ {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
    ,
    { "ISBN":"ISBN-0-13-815504-6",
      "Price":100,
      "Remark":"Buy this book bundled with 'A First Course' - a great deal!",
      "Title":"Database Systems:The Complete Book",
      "Authors":[ {"First_Name":"Hector", "Last_Name":"Garcia-Molina"},
                  {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
  ]
```

JavaScript Object Notation (JSON)

- No longer tied to JavaScript
- Parsers for many languages

```
{ "Books":
  [
    { "ISBN":"ISBN-0-13-713526-2",
      "Price":85,
      "Edition":3,
      "Title":"A First Course in Database Systems",
      "Authors":[ {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
    ,
    { "ISBN":"ISBN-0-13-815504-6",
      "Price":100,
      "Remark":"Buy this book bundled with 'A First Course' - a great deal!",
      "Title":"Database Systems:The Complete Book",
      "Authors":[ {"First_Name":"Hector", "Last_Name":"Garcia-Molina"},
                  {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
```

```
{ "Books":
  [
    { "ISBN":"ISBN-0-13-713526-2",
      "Price":85,
      "Edition":3,
      "Title":"A First Course in Database Systems",
      "Authors":[ {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
    ,
    { "ISBN":"ISBN-0-13-815504-6",
      "Price":100,
      "Remark":"Buy this book bundled with 'A First Course' - a great deal!",
      "Title":"Database Systems:The Complete Book",
      "Authors":[ {"First_Name":"Hector", "Last_Name":"Garcia-Molina"},
                  {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
  ],
  "Magazines":
  [
    { "Title":"National Geographic",
      "Month":"January",
      "Year":2009 }
    ,
    { "Title":"Newsweek",
      "Month":"February",
      "Year":2009 }
  ]
}
```

Basic constructs (recursive)

- Base values
  number, string, boolean, …
- Objects { }
  sets of label-value pairs
- Arrays [ ]
  lists of values

Relational Model versus JSON

	Relational	JSON
Structure	Tables	Nested Sets & Arrays
Schema	Fixed in advance	"self-describing" – more flexible
Queries	Simple expressing languages	Nothing widely used
Ordering	None	Arrays (ordered)
Implementation	Native systems	Coupled with PLs and NoSQL systems

XML versus JSON

	XML	JSON
Verbosity	More	Less
Complexity	More	Less
Validity	DTDs, XSDs – widely used	JSON schema – not widely used
Prog. Interface	Clunky – "impedance mismatch"	More direct
Querying	Xpath, Xquery, XSLT	JSON Path, JSON Query, JAQL

BSON

- Like JSON, BSON supports the embedding of documents and arrays within other documents and arrays. BSON also contains extensions that allow representation of data types that are not part of the JSON spec. For example, BSON has a *Date* type and a *BinData* type.
- The driver performs translation from the language's "object" (ordered associative array) data representation to BSON, and back.

BSON

- Using BSON (binary JSON), developers can easily map to modern object-oriented languages without a complicated ORM (object-relational mapping) layer.
- Is a binary encoded serialization of JSON-like documents; zero or more key/value pairs are stored as a single entity.
- Lightweight, traversable, efficient

```
{"hello": "world"}                  →  "\x16\x00\x00\x00\x02hello\x00
                                        \x06\x00\x00\x00world\x00\x00"

                                       "1\x00\x00\x00\x04BSON\x00&\x00
                                        \x00\x00\x020\x00\x08\x00\x00
{"BSON": ["awesome", 5.05, 1986]}   →   \x00awesome\x00\x011\x00333333
                                        \x14@\x102\x00\xc2\x07\x00\x00
                                        \x00\x00"
```

http://bsonspec.org/#/specification

mongoDB

MongoDB Overview

- From "hu*mongo*us"
- Document-oriented database, not relational
- Data serialized to BSON
- Manages hierarchical collection of BSON documents
- Schema free
- Written in C++
- Has an official driver for C# with support from 10gen
- Scalable with high-performance (scales horizontally)
- Designed to address today's workloads
- Runs nearly everywhere
- BASE rather than ACID compliant
- Replication
- Part of the "NoSQL" class of DBMS
- Website with list of all features - http://www.mongodb.org/

In Production

http://www.mongodb.org/about/production-deployments/

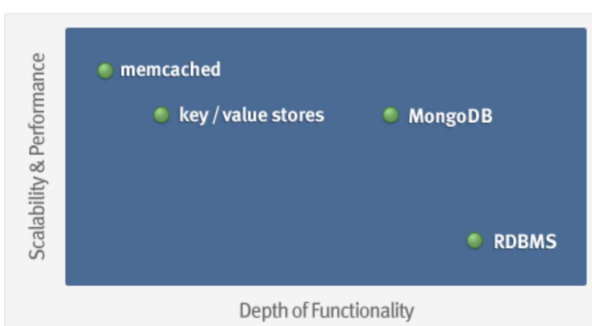
Example Use Cases

Archiving – Craigslist
Content management – MTV Networks
Commerce – CustomInk
Gaming – Disney
Real-time analytics – Intuit
Social networking - foursquare

Supported languages

MongoDB Goal

- **Goal:** bridge the gap between key-value stores (which are fast and scalable) and relational databases (which have rich functionality).



What is MongoDB?

- MongoDB is an open source, document-oriented database designed with both scalability and developer agility in mind. Instead of storing your data in tables and rows as you would with a relational database, in MongoDB you store JSON-like documents (specifically, BSON) with dynamic schemas (schema-free, schemaless).

Data Model

1. Mongo system: Collection of databases.
2. Database: Set of collections.
3. Collection: Set of documents.
4. Document: Set of fields.
5. Field: Key-Value pair.
   - Key : Name (String).
   - Value : String, integer, float, timestamp, binary, document, array of values.

http://hemantblogs.blogspot.com/2012/07/introduction-to-mongo-db.html

RDBMS vs MongoDB

RDBMS		MongoDB
Database	⇨	Database
Table, View	⇨	Collection
Row	⇨	Document (JSON, BSON)
Column	⇨	Field
Index	⇨	Index
Join	⇨	Embedded Document
Foreign Key	⇨	Reference
Partition	⇨	Shard

Querying: CRUD

- Create
  - db.collection.insert(<document>)
  - db.collection.save(<document>)
  - db.collection.update(<query>, <update>, { upsert: true })
- Read
  - db.collection.find(<query>, <projection>)
  - db.collection.findOne(<query>, <projection>)
- Update
  - db.collection.update(<query>, <update>, <options>)
- Delete
  - db.collection.remove(<query>, <justOne>)

CRUD example

```
> db.user.insert({
      first: "John",
      last : "Doe",
      age: 39
})
```

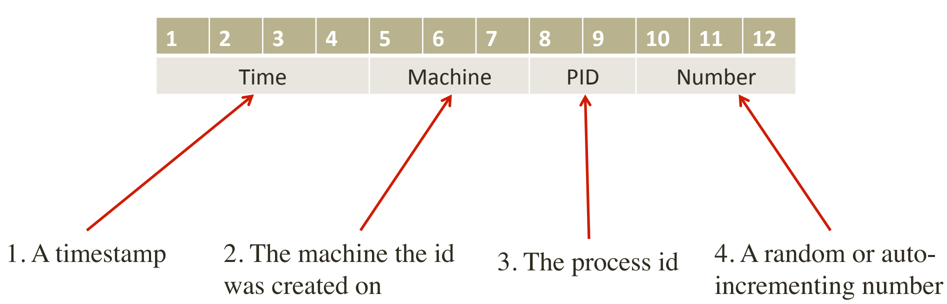
```
> db.user.find ()
{
      "_id" : ObjectId("51…"),
      "first" : "John",
      "last" : "Doe",
      "age" : 39
}
```

```
> db.user.update(
      {"_id" : ObjectId("51…")},
      {
            $set: {
                  age: 40,
                  salary: 7000}
      }
)
```

```
> db.user.remove({
      "first": /^J/
})
```

The Mongo ID

The Mongo ID is a unique id automatically generated for each document created. The id is 12 byte value that is composed of several factors.



MongoDB - Querying

Share

- find(): Returns a cursor containing a number of documents
  - All users
    - db.users.find()
  - User with id 42
    - db.users.find({ _id: 42})
  - Age between 20 and 30
    - db.users.find({ age: { $gt: 20, $lt: 30 } })
  - Subdocuments: ZIP 5026
    - db.users.find({ address.zip: 5026 })
  - OR: ruben or younger than 30
    - db.users.find({ $or: [
      { name : "ruben" },
      { age: { $lt: 30 } }
      ]})
  - Projection: Deliver only name and age
    - db.users.find({ }, { name: 1, age: 1 })

```
{
   "_id": 42,
   "name": "ruben",
   "surname": "inoto",
   „age": „36",
   "address": {
      "street": "Glaserstraße",
      "zip": "5026" }
}
```

Embed and Link

- Embedding is the nesting of objects and arrays inside a BSON document.
- Links are references between documents.
- There are no joins in MongoDB – distributed joins would be difficult on a 1,000 server cluster.
- Embedding is a bit like "prejoined" data. Operations within a document are easy for the server to handle.
- Links must be processed at client-side by the application; the application does this by issuing a follow-up query.
  - save the _id field of one document in another document as a reference. Then your application can run a second query to return the linked data. These references are simple and sufficient for most use cases.
- Generally, for "contains" relationships between entities, embedding should be chosen. Use linking when not using linking would result in duplication of data.

More detail on referencing:
http://www.mongodb.org/display/DOCS/Database+References

Link Example

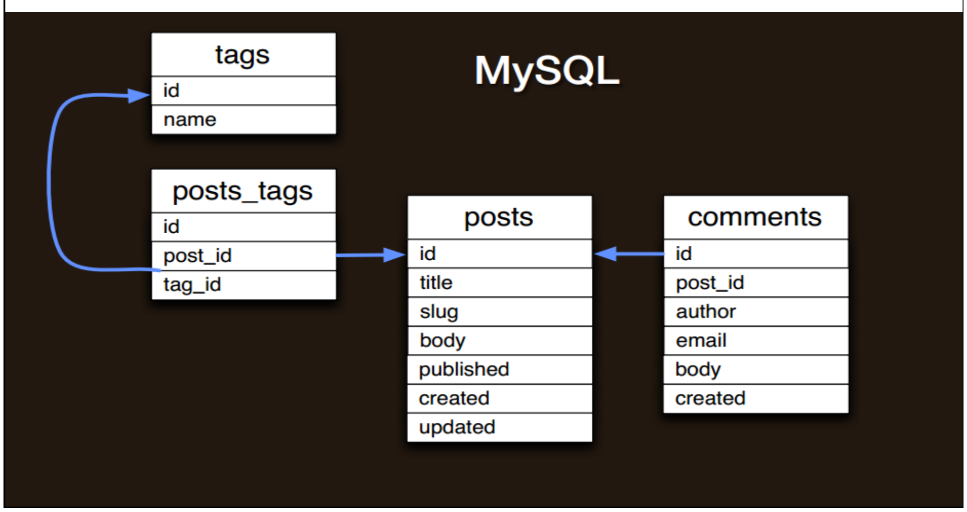
- Consider the following operation to insert two documents, using the **_id field of the first document as a reference in the second document:**

```
original_id = ObjectId()

db.places.insert({
    "_id": original_id
    "name": "Broadway Center"
    "url": "bc.example.net"
})

db.people.insert({
    "name": "Erin"
    "places_id": original_id
    "url":  "bc.example.net/Erin"
})
```

- Then, when a query returns the document from the **people collection you can, if needed, make a second query for the document referenced by the places_id field in the places collection.**

http://docs.mongodb.org/manual/reference/database-references/

RDBMS Schema Design



MongoDB Schema Design

MongoDB

Schema Design

MongoDB

```
{
    "_id" : ObjectId("4c03e856e258c2701930c091"),
    "title" : "Welcome to MongoDB",
    "slug" : "welcome-to-mongodb",
    "body" : "Today, we're gonna totally rock your world...",
    "published" : true,
    "created" : "Mon May 31 2010 12:48:22 GMT-0400 (EDT)",
    "updated" : "Mon May 31 2010 12:48:22 GMT-0400 (EDT)",
    "comments" : [
        {
            "author" : "Bob",
            "email" : "bob@example.com",
            "body" : "My mind has been totally blown!",
            "created" : "Mon May 31 2010 12:48:22 GMT-0400 (EDT)"
        }
    ],
    "tags" : [
        "databases", "MongoDB", "awesome"
    ]
}
```

MongoDB Summary

- Document-Oriented storege
- Full Index Support
- Replication & High Availability
- Auto-Sharding
- Querying
- Fast In-Place Updates
- Map/Reduce

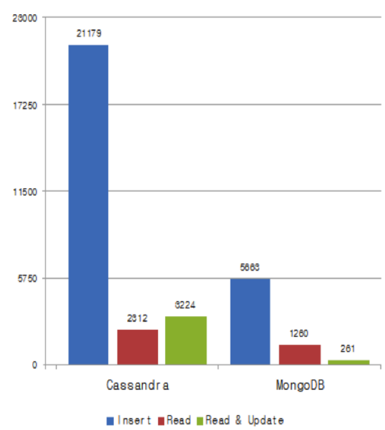
Agile

Scalable

Cassandra vs MongoDB Comparison

Cassandra is:

Nearly 4x better in writes
Nearly 2x better in reads
Over 12x better in reads/updates



YCSB Benchmark
Source: http://blog.cubrid.org/dev-platform/nosql-benchmarking/?utm_source=NoSQL+Weekly+List&utm_campaign=143fae86b2-NoSQL_Weekly_Issue_41_September_8_2011&utm_medium=email