EE 542
Lecture 15: Database in the Cloud

Internet and Cloud Computing

Young Cho
Department of Electrical Engineering
University of Southern California

Main Use: Big Data Processing

- Information Abstraction
  - Summarization
  - Semantics Analysis
  - Pattern matching
  - MapReduce
- Information Partitioning
  - Data cleansing
  - Indexing and Organization
  - K-means Clustering

Data Storage and Management

- Database
  - Organized collection of data
  - Operate large quantities of information
- Database Management System (DBMS)
  - Software to Creation and Maintain Database
  - Organze, Store, and Retrieve data

History of Database

- 1960's: Hierarchical File-based
- 1970's: Networked and Relational
- 1980's: Entity-Relationship
- 1990's: Object Oriented Server-Clients
- 2000's: Web-based Server-Clients
- Current: Cloud-base with Clients

Relational Database (RDB)

- Designed for all purposes
- Strong consistancy, concurrency, recovery
- Mathematical background
- Standard Query language (SQL)

ACID Semantics

- **A**tomicity: All or nothing.
- **C**onsistency: Consistent state of data and transactions.
- **I**solation: Transactions are isolated from each other.
- **D**urability: When the transaction is committed, state will be durable.

Any data store can achieve Atomicity, Isolation and Durability but do you always need consistency? No.

By giving up ACID properties, one can achieve higher performance and scalability.

Scaling Up the Database

- RDBs Do Not Scale
  - Hard to scale horizontally
  - Does not conceptually map to multiple nodes
- Expensive
  - Expensive hardware costs
  - Expensive hardware maintenance
- Poor Result
  - Low Speed (performance)
  - Low availability
  - Does not tolerate Partitioning

Current/Emerging Data Needs

2002
2004
2006
2008
2010
2012
Big data
Connectivity
P2P Knowledge
Concurrency
Diversity
Cloud-Grid

Consistency

- Clients should read the same data.
- Levels of Consistency
  - Strict Consistency – RDBMS
  - Tunable Consistency – Cassandra
  - Eventual Consistency – Amazon Dynamo
- Client perceives atomic operation

BASE, an ACID Alternative

- Basically Available
  - Nodes in the a distributed
  - Individual node may go down
  - Whole system still functional with the rest
- Soft State
  - System state and data changes over time
- Eventual Consistency
  - Given enough time, data will be consistent across the distributed system.

Different Priorities

- ACID
  - Strong consistency
  - Less availability
  - Pessimistic concurrency
  - Complex
- BASE
  - Availability is the most important thing
  - Weaker consistency (Eventual)
  - Best effort
  - Simple and fast
  - Optimistic

Emergence of NoSQL

- Not only SQL
  - Does not follow all of the SQL requirements
  - However, provides SQL-like interface
  - Scalability through Relaxed models
- Removes
  - Overhead of SQL constrained transactions
  - Complexity of SQL query
  - Necessity of up-front schema design
- Allows
  - Easy and frequent changes to DB
  - Fast development
  - Higher Scalability

Not Good for Everything

- Financial Data
- Commercial Transactions
- Business Critical Data

Still Useful for Lots

Google

ebay

YAHOO!

LinkedIn

NETFLIX

amazon

theguardian

NoSQL

- NoSQL databases reject:
  - Overhead of ACID transactions
  - “Complexity” of SQL
  - Burden of up-front schema design
  - Declarative query expression
  - Yesterday’s technology
- Programmer responsible for
  - Step-by-step procedural language
  - Navigating access path

NoSQL Database Types

NoSQL databases is complicated because there are a variety of types:

- Key-Value Store – Hash table of keys
- Column Store – Each storage block contains data from only one column
- Document Store – stores documents made up of tagged elements
- Graph Databases

SQL vs NoSQL

- SQL Databases
  - Predefined Schema
  - Standard definition and interface language
  - Tight consistency
  - Well defined semantics
- NoSQL Database
  - No predefined Schema
  - Per-product definition and interface language
  - Getting an answer quickly is more important than getting a correct answer

Examples: Key-Value Stores

- MapReduce
  - Values stored with Keys
  - Fast access to small data values
- Example
  - Project-Voldemort - http://www.project-voldemort.com/ (Linkedin)
  - MemCacheDB - http://memcachedb.org/

Example: Document Database

- Schema Free
- JSON – JavaScript Object Notation
- Query Model: JavaScript or custom
- Aggregations: Map/Reduce
- Indexes are done via B-Trees
- Example
  - CouchDB - http://couchdb.apache.org/ (BBC)
  - MongoDB - http://www.mongodb.org/ (Foursquare, Shutterfly)

CouchDB JSON Tags

- "_id"
  - GUID – Global Unique Identifier
  - Passed in or generated by CouchDB
- "_rev"
  - Revision number
  - Versioning mechanism
- "type", "author", "title", etc.
  - Arbitrary tags
  - Schema-less
  - Could be validated after the fact by user-written routine

MongoDB

- Map Reduce with JavaScript - Aggregation
- You have indexes
  - B-Trees. Ids are always indexed.
- Updates are atomic
  - Low contention locks
- Querying mongo done with a document
- Repository Pattern

Example: Graph Stores

- Based on Graph Theory.
- Scale vertically, no clustering.
- You can use graph algorithms easily.

Example: HBase

- HDFS provides us with a filesystem consisting of arbitrarily large files that can only be written once and are should be read sequentially, end to end.
- Online transactions want to read and write individual cells in a large table. (e.g. update inventory and price as orders come in.)
- HBase implements online transaction on top of HDFS by using additional storage and memory to organize the tables, and writing them back to HDFS as needed.
- HBase is an open source implementation of the BigTable design published by Google.

HBase Data Model

- A database consists of multiple tables.
- Each table consists of multiple rows, sorted by row key.
- Each row contains a row key and one or more column families.
- Each column family is defined when the table is created.
- Column families can contain multiple columns. (family:column)
- A cell is uniquely identified by (table,row,family:column).
- A cell contains an uninterpreted array of bytes and a timestamp.

Data in Tabular Form

	Name		Home		Office	
Key	**First**	**Last**	**Phone**	**Email**	**Phone**	**Email**
101	Florian	Krepsbach	555-1212	florian@ wobegon.org	666-1212	fk@phc.com
102	Marilyn	Tollerud	555-1213		666-1213	
103	Pastor	Inqvist			555-1214	inqvist@ wels.org

Data in Tabular Form

	Name			Home		Office		Social
Key	First		Last	Phone	Email	Phone	Email	FacebookID
101	Florian	Garfield	Krepsbach	555-1212	florian@wobegon.org	666-1212	fk@phc.com	
102	Marilyn		Tollerud	555-1213		666-1213		
103	Pastor		Inqvist			555-1214	inqvist@wels.org	

New columns can be added at runtime.

Column families cannot be added at runtime.

Nested Data Representation

```
Table People ( Name, Home, Office )
{
        101: {
                Timestamp: T403;
                Name: {First=“Florian”, Middle=“Garfield”, Last=“Krepsbach”},
                Home: {Phone=“555-1212”, Email=“florian@wobegon.org”},
                Office: {Phone=“666-1212”, Email=“fk@phc.com”}
        },
        102: {
                Timestamp: T593;
                Name: { First=“Marilyn”, Last=“Tollerud”},
                Home: { Phone=“555-1213” },
                Office: { Phone=“666-1213” }
        },
        …
}
```

Nested Data Representation

```
GET People:101
          101: {
                    Timestamp: T403;
                    Name: {First="Florian", Middle="Garfield", Last="Krepsbach"},
                    Home: {Phone="555-1212", Email="florian@wobegon.org"},
                    Office: {Phone="666-1212", Email="fk@phc.com"}
          }

GET People:101:Name
          People:101:Name: {First="Florian", Middle="Garfield", Last="Krepsbach"}

GET People:101:Name:First
          People:101:Name:First="Florian"
```

Fundamental Operations

- CREATE table, families
- PUT table, rowid, family:column, value
- PUT table, rowid, whole-row
- GET table, rowid
- SCAN table *(WITH filters)*
- DROP table

Relaxed ACID Requirements

- Atomicity - Entire rows are updated atomically or not at all.
- Consistency:
  - A GET is guaranteed to return a complete row that existed at some point in the table's history. (Check the timestamp to be sure!)
  - A SCAN must include all data written prior to the scan, and may include updates since it started.
- Isolation: Not guaranteed outside a single row.
- Durability: All successful writes have been made durable on disk.