Data Management in the Cloud

Data Models, Application Characteristics
(Lecture 2)

1

Let's Remember…

- Cloud Computing
  - Utility Computing
  - Virtualization
  - Economics (pay as you go)
- Data management in the cloud
  - Cloud characteristics (elasticity if parallelizable, untrusted host, large distances)
  - Transactional vs. Analytical
  - Wish List
- DB vs. NoSQL in two lines…
  - Database: complex / concurrent
  - NoSQL: simple / scalable

2

Data Management in the Cloud (Lecture 2)

SCALABLE DATA STORES

Overview

- New systems have emerged to address requirements of data management in the cloud
  - so-called “NoSQL” data stores
  - scalable SQL databases
- **Horizontal Scalability**
  - shared nothing
  - replicating and partitioning data over thousands of servers
  - distribute “simple operation” workload over thousands of servers
- **Simple Operations**
  - key lookups
  - read and writes of one or a small number of records
  - **no** complex queries or joins

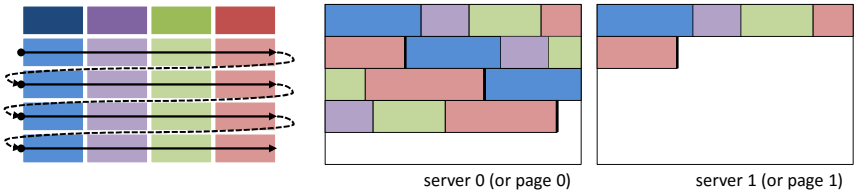
Partitioning/Sharding

- Horizontal Partitioning / Sharding
  - Store records on different servers
- Vertical Partitioning
  - Store parts of a single record on different servers (or maybe in different files)

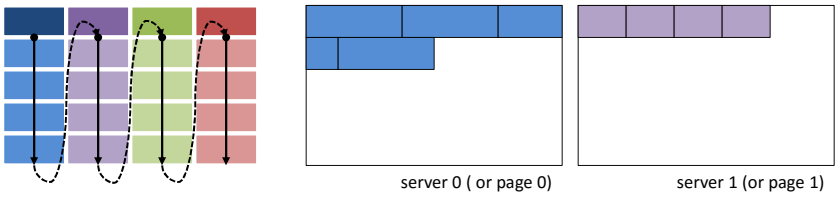
5

Horizontal vs. Vertical Partitioning

- Horizontal Partitioning / Sharding / **row-wise** order



- Vertical Partitioning



6

Slide Credit: Torsten Grust, University of Tübingen, Germany

Defining "NoSQL"

- No agreed upon definition
  - "not only SQL"
  - "not relational"
  - …
- Six key features, according to Cattell
  1. ability to horizontally scale simple operation throughput over many servers
  2. ability to replicate and distribute (partition) data over many servers
  3. simple call level interface or protocol (in contrast to an SQL binding)
  4. weaker concurrency model than ACID transactions of most relational (SQL) database systems
  5. efficient use of distributed indexes and RAM for data storage
  6. ability to dynamically add new attributes to data records

Based on: "Scalable SQL and NoSQL Data Stores" by R. Cattell, 2010

7

Data Models

- Terminology
  - **tuple:** row in a relational table, where attribute names and types are defined by a schema, and values must be scalar
  - **document:** supports both scalar values and nested documents, and the attributes are dynamically defined for each document
  - **column family:** groups key/value pairs (columns) into families to partition and replicate them; one column family is similar to a document as new (nested, list-valued) attributes can be added
  - **object:** analogous to objects in programming languages, but without procedural methods
- Relational
  - data is stored in relations (tables) of tuples (rows) of scalar values
  - queries expressed over arbitrary (combinations of) attributes
  - indexes defined over arbitrary (combinations of) attributes

Based on: "Scalable SQL and NoSQL Data Stores" by R. Cattell, 2010

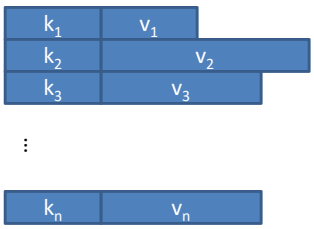
8

Outline: Data Models

- No-SQL (in order of complexity)
  - Key-Value (Voldemort, Riak)
  - Document (SimpleDB, CouchDB)
  - Column-Family/Extensible Record Stores (BigTable, Hbase, Cassandra)
- Other
  - Graph (Neo4j)
  - Array (SciDB)
  - OODB (Orleans)
  - Scalable Relational (Volt DB)

9

Key/Value Data Model



- Interface
  - put(key, value)
  - get(key): value

- Data storage
  - values (data) are stored based on programmer-defined keys
  - system is agnostic as to the structure (semantics) of the value
- Queries are expressed in terms of keys
- Indexes are defined over keys
  - Typically primary only

10

Key/Value Systems II

- Project Voldemort (LinkedIn) (2009)
  - Updates replicas asynchronously – no guarantee of consistent data (MVCC)
  - Can guarantee up-to-date if you read a majority of replicas
  - Optimistic locking for multi-record updates
  - Supports automatic sharding
  - Automatically detects and recovers failed nodes, automatically adapts to added/removed notes
- Recent additions
  - Alternative storage engines
  - Alternative serializers
  - Hadoop → Voldemort bulk transfer of read-only data

11

Key/Value Systems II

- Riak (2009)
  - Open-source version of Amazon Dynamo DB
  - "Advanced" key-value store
  - Objects fetched / stored in JSON
  - Lookup and indices only on primary key
  - Sharding on primary key
  - Symmetric architecture, no distinguished node
- Recent Additions
  - New datatypes: sets, maps, flags
  - Write-once data

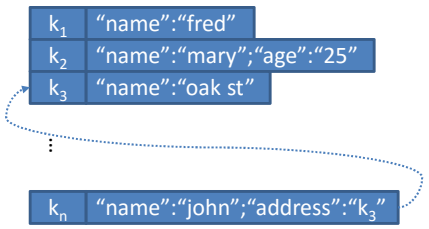
12

Key/Value Systems - Summary

- Simple key-value lookups ("value" is opaque to system)
- Scalability through key distribution over nodes
- Usually persistence plus some of : replication, versioning, locking, transactions, etc…
- Interface: insert, delete, lookup
- Consistency is usually per key
- Voldemort/Riak – MVCC, Redis, Scalaris, Mem* use locking…
- Use Case
  - Simple application, one kind of object – look up on one attribute
    For example, looking up user profiles based on user id
  - Simple, easy to use

13

Document Data Model



- Interface
  - set(key, document)
  - get(key): document
  - set(key, name, value)
  - get(key, name): value

- Data storage
  - documents (data) is stored based on programmer-defined keys
  - system is aware of the (arbitrary) document structure
  - support for lists and nested documents
- Queries expressed in terms of key (or attribute, if index exists)
- Support for key-based indexes and secondary indexes

14

Document Systems

- More complex data than key-value stores
- Schema: Collection (group of documents) + Attributes (name)
- What is a "document"?
  - Can store Microsoft Word documents, but "document" is more general than that
  - "Document" means a "pointerless object" with structure
- Usually support secondary indices
- Usually multiple types of documents per database
- Usually nested documents
- No ACID transaction properties across documents
- SimpleDB, CouchDB, MongoDB

15

Document Systems II

- SimpleDB (Amazon) (2007)
  - Select, Delete, GetAttr, PutAttr
  - Does not allow nested documents
  - Supports eventual consistency
  - Asynchronous replication
  - Multiple Domains within a database
  - Domains support multiple indices
  - Indices automatically updated when document attrs modified
  - Does not automatically partition (shard) data
- Recent additions
  - Conditional put and delete
  - Consistent read of your own writes
  - DynamoDB is successor

16

Document Systems III

- CouchDB (Apache) (2008)
  - Schema is based on collections
  - Secondary indices must be explicitly created
  - Queries are in Javascript
  - Indices are B-Trees
  - Queries as Javascript => burden on users
  - Asynchronous replication for scalability (not sharding)
  - Queries are map-reduce views
- Recent additions
  - Clusters: CouchDB instance spread over several machines
  - Mango query language

17

Document Systems IV

- MongoDB (10gen) (2009)
  - Supports automatic sharding
  - Replication primarily for failover
  - Dynamic queries with automatic use of indices
- Recent additions
  - Capped collections
  - Alternative storage engines
  - Recursive & faceted search
  - Read-only views

18

Document Systems - Summary

- Schema-less except for attributes
- Provide a mechanism to query collections based on multiple attribute constraints
- Typically no explicit locks, weaker concurrency and atomicity than traditional ACID databases
- Documents can be distributed over all nodes, but scalability differs
  - Replication (all)
  - Sharding (Automatic in MongoDB)
- Use case
  - Multiple different kinds of objects
  - Need to look up based on multiple fields
  - Level of concurrency? Can you tolerate eventual consistency?

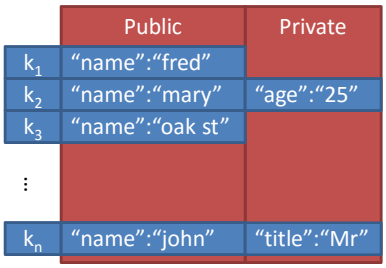
19

Key / Value vs. Document

- In Key / Value databases, the “value” (“aggregate” in the book) is opaque to the database
- In a Document database, the database can see the structure of the “value” (“aggregate”)
- Key/Value – lookup based on key only
- Document – can look up based on structure inside the document
- Line is a bit blurry…

20

Column Family Data Model



- Interface
  - define(family)
  - insert(family, key, columns)
  - get(family, key): columns

- Data storage
  - <name, value, timestamp> triples (so-called columns) are stored based on a column family and key; a column family is similar to a document
  - system is aware of (arbitrary) structure of column family
  - system uses column family information to replicate and distribute data
- Queries are expressed based on key and column family
- Secondary indexes per column family are typically supported

21

Column Family Systems I

BigTable (Google) (Started 2004)

- Scalability: splitting rows and columns over multiple nodes
  - Horizontal and vertical partitioning
- Rows split over nodes through sharding on primary key
  - Typically split by range and not hash
- Columns distributed over multiple nodes by using "column groups"
  - Customer indicates which Columns are best stored together
- Most extensible-record stores patterned on BigTable
- HBase is open-source version (Apache) of BigTable

22

Column Family Systems II

- Cassandra (Facebook) (2008)
  - Ordered hash index
  - Weak concurrency model, no locking, asynchronously updated replicas
  - Adds concept of "super column" – similar to column groups in BigTable
- Recent additions
  - Secondary indexes
  - Cassandra query language
  - Triggers

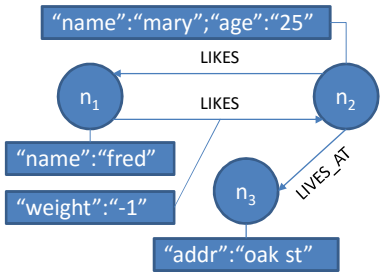
23

Column Family Summary

- Use cases similar to Document Stores
  - Multiple kinds of objects
  - Lookups based on any field
  - Typically higher throughput
  - Typically stronger concurrency guarantees
  - Support for vertical partitioning

24

Graph Data Model



- Interface
  - create: id
  - get(id)
  - connect(id_1, id_2): id
  - addAttribute(id, name, value)
  - getAttribute(id, name): value

- Data storage
  - data is stored in terms of nodes and (typed) edges
  - both nodes and edges can have (arbitrary) attributes
- Queries are expressed based on system ids (if no indexes exist)
- Secondary indexes for nodes and edges are supported
  - retrieve nodes by attributes and edges by type, start and/or end node, and/or attributes

25

Graph Query Language

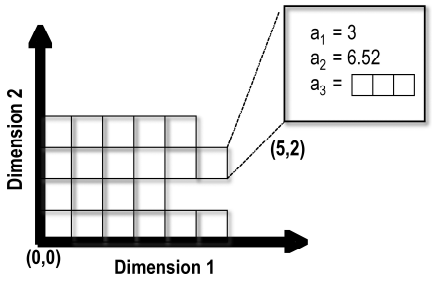
- Support for navigational access: Cypher (Neo4J)

```
MATCH
(A:Person)-[:LIKES]->(B:Person)
    -[:LIVES_AT]->(C)
WHERE B.age = 25
RETURN A.Name, C.Addr
```

26

Array Data Model

- Nested multi-dimensional arrays
  - cells can be tuples or other arrays
  - can have non-integer dimensions
- Additional "History" dimension on updatable arrays
- Ragged arrays allow each row or column to have a different length
- Supports multiple flavors of "null"
  - array cells can be "EMPTY"
  - user-definable treatment of special values



27

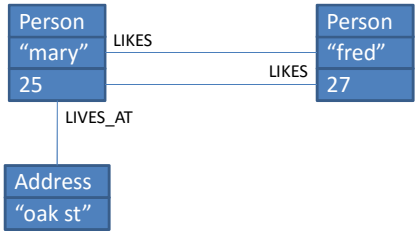
SciDB Data Definition

```
CREATE ARRAY Test_Array
     < A: integer NULLS,
       B: double,
       C: USER_DEFINED_TYPE >
      [I=0:99999,1000, 10, J=0:99999,1000, 10 ]
      PARTITION OVER ( Node1, Node2, Node3 )
      USING block_cyclic();
```

Attribute names	A, B, C
Index names	I, J
Chunk size	1000
Overlap	10

28

Object Data Model



- Interface
  - set(object)
  - get(query): object

- Data storage
  - typed programming language objects (plus referenced objects) stored
  - attribute can be collection-valued
  - database is aware of the type (schema) of objects
- Objects are retrieved using queries or by traversal from "roots"
- Specialized indexes can be expressed based on schema

29

Object-Oriented System

- Orleans (Microsoft) (2010, open source 2015)
  - Distributed in-memory object database (with optional persistence)
  - *Virtual-actor model*: Objects are activated on demand
  - Asynchronous messaging, single-threaded objects
  - Fault tolerance, load balancing
  - Used for Halo 4 & 5
  - Transactions (new)

30

Scalable Relational Systems

- Complete pre-defined schema
- SQL Interface
- ACID transactions
- Scalability may be comparable to NoSQL data stores given:
  - Use small-scope operations
  - Use small-scope transactions
  - NoSQL systems avoid these issues by making it impossible to perform large-scope operations and transactions
- Good if:
  - Complex schema
  - Programmers familiar with SQL
  - You need/want ACID transactions (what is your concurrent access pattern?)
  - Tools!

31

Scalable Relational Systems

- MySQL Cluster (2004)
  - One of the earliest scalable relational systems
  - Shared-nothing architecture
- Volt DB (2010)
  - Tables partitioned over multiple servers
  - Shared-nothing architecture
  - Customer can chose sharding attribute
  - Selected tables can be replicated (read-mostly data)
  - Designed for a database that fits in (distributed) RAM
  - Indices/record structures desinged for RAM (not disk)
  - All access through stored procedures – no user interaction
  - High throughput, not necessarily low latency

32

References

- M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, M. Zaharia: **Above the Clouds: A Berkeley View of Cloud Computing.** *Tech. Rep. No. UCB/EECS-2009-28, 2009.*
- D. J. Abadi: **Data Management in the Cloud: Limitations and Opportunities**. *IEEE Data Eng. Bull. 32(1), pp. 3—12, 2009.*
- R. Agrawal, A. Ailamaki, P. A. Bernstein, E. A. Brewer, M. J. Carey, S. Chaudhuri, A. Doan, D. Florescu, M. J. Franklin, H. Garcia- Molina, J. Gehrke, L. Gruenwald, L. M. Haas, A. Y. Halevy, J. M. Hellerstein, Y. E. Ioannidis, H. F. Korth, D. Kossmann, S. Madden, R. Magoulas, B. Chin Ooi, T. O'Reilly, R. Ramakrishnan, S. Sarawagi, M. Stonebraker, A. S. Szalay, G. Weikum: **The Claremont Report on Database Research.** 2008.
- R. Cattell: **Scalable SQL and NoSQL Data Stores.** *SIGMOD Rec. 39(4), pp. 12—27, 2010.*

33

Data Management in the Cloud (Lecture 2)

APPLICATION CHARACTERISTICS

34

Application Characteristics

Aspects of applications that influence the choice of data management system

- Inward facing vs. outward facing
- How critical is consistency?
- Do answers need to be precise
  - Accurate data
  - Complete computation
- Interactive vs. data analysis?
  - What kind of response time is needed?
- Does the data partition easily?

35

Application Characteristics 2

- How complex is the data?
  - flat records
  - files (e.g., audio)
  - nested structure
- How much heterogeneity in the data?
- How much data is there?
- What is the update pattern?
- How sensitive and valuable is the data?
- How much variability in demand?

36

Application Characteristics 3

- How complex is the logic?
- What is the data access pattern?
  - Are there hot spots and cold spots?
  - Is there locality of access?
- How complex are the access patterns for common operations?
  - Single data item?
  - Multiple data items?
  - Large chunks of the data?
- Can the data in use fit in memory?

37