NoSQL (or Not Only SQL) Systems

Sharma Chakravarthy
Information Technology Laboratory
Computer Science and Engineering Department
The University of Texas at Arlington, Arlington, TX 76009
Email: sharma@cse.uta.edu
URL: http://itlab.uta.edu/sharma

Acknowledgements
- These slides are put together from a variety of sources (both papers and slides/tutorials available on the web)
- Mostly I have tried to: provide my perspective, emphasize aspects that are of interest to this course, and have tried to put forth a consolidated view of Cloud computing

Tutorial Outline
- Before the cloud
- Cloud Definitions, trend
- Cloud computing characteristics
- Differences between cc, grid, and other forms of computing
- Cloud Architecture
- Map/reduce
- Hadoop
- Map/Reduce vs. DBMS
- NoSQL
- Conclusions

RDBMSs
- Need Schema
  - EER modeling, normalization, and generation of schema
  - Schema is flat
  - Schema difficult to change
  - No nesting of tables!
- Do lots of joins
  - Joins are expensive, are quite well optimized
  - Normalization increases number of joins
- Support limited Data types
  - Text and numeric data types
  - Not geared for others (voice, image, logs, graphs, objects etc.)
  - Lacks general search capability (e.g., find all occurrences in the database)
RDMSs

- Were optimized
  - For joins (not so much for self-joins! Also for small number of joins)
  - Fast retrieval
  - For management of data over a long period of time
  - Generating reports
  - OLTP
  - Warehouses for OLAP
  - High concurrency
  - Recovery
  - ACID properties
- SQL sees all data as a single data set!
  - Partitioning is not automatic
  - Query plan takes advantage of partitioning

Newer applications

- Do not exactly fit the above model
  - No schema (or loose schema or need schema flexibility)
  - One shot computation
  - No concurrent usage
  - Recovery is of a different nature (fault tolerance instead of strict consistency)
  - Custom computation; difficult to express in SQL
  - Needs custom optimization rather than general-purpose optimization
  - Data sizes beyond what DBMSs could handle
  - Quick turn-around time
  - ACID properties were not needed as in an RDBMS
- Answer: NoSQL systems
  - Data model, Customizable ACID properties

An example

- Suppose I want to keep track of customer orders, items in each order, how each order was paid.
- This is a complex objects that looks like:

  **Objects representation**
  
<table>
<thead>
<tr>
<th>Order ID: 1001</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer: mary</td>
</tr>
<tr>
<td>Line items:</td>
</tr>
<tr>
<td>item 1 qty per item total;</td>
</tr>
<tr>
<td>item 2 qty per item total;</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

  **Payment info:**
  
  cc: visa
  cc #: exp date:

  **Relational representation**
  
<table>
<thead>
<tr>
<th>Orders table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customers table</td>
</tr>
<tr>
<td>Items table</td>
</tr>
<tr>
<td>Credit_info table</td>
</tr>
</tbody>
</table>

  To put together an object, you have to do multiple joins on the above tables!

NoSQL Databases

- Google came up with BigTable as an alternative to using RDBMS
- Amazon came up with Dynamo
- These did not use SQL (write your own code)
- The term was conjured up as a hashtag (#nosql) for a meeting!
- Initial group of users
  - MongoDB
  - couchDB
  - HyperTable
  - Cassandra
  - Dynamo
  - Apache Hbase, voldemort,
Definition of a NoSQL Database

- It is difficult to give a definition as there is no consensus
- Hence, characteristics of NoSQL databases
  - Non-relational (uses different data models)
  - Primarily open-source
  - Schema less (or loose schema)
  - Elasticity (both storage and server capacity can be added on-the-fly)
  - Sharding (non-monolithic data representation)
  - Asynchronous replication (not raid)
  - BASE (Basically Available, Soft state, Eventual consistency) and CAP (Consistency, Availability, and Partition tolerance) instead of ACID
- If you think about hierarchical and network databases, they will qualify as NoSQL, but lacked some of the characteristics mentioned above!

Data Models

- **Documents**
  - mongoDB
  - couchDB
  - Raven DB
- **Column-family**
  - Cassandra
  - Apache Hbase, HyperTable
- **Key-value**
  - Project Voldemort
  - Riak
  - Redis
- **Graph**
  - Neo4j, GraphDB

Key-Value store

- Value can be anything you want (single number to complex object to an image)
  - String, json object, BLOB etc.
  - See how this will keep all information about an order together with order id.
  - You can think of key-value store as a hash map!

Key-Value store (2)

- A hash representation is used
- Caching is used to enhance performance
- Operations
  - Get(key)
  - Put(key)
  - Multi-get(key1, key2, ..., keyn)
  - Delete(key)
- As data size increases, additional performance issues come up
  - Keeping keys Unique may be difficult
- Examples: Oracle NoSQL, Riak and amazon Dynamo
**Key-Value store (3)**

- You can envision how partitioning and scalability is achieved.
- You can partition data and store separately.
- If the keys are unique, updates can be processed in parallel and happens only in one partition.
- Scalability comes from hash-based approach and simplicity of management.
- May be good for some applications!

**Document store (2)**

- No schema or flexible schema.
- Query or update portions of the document structure.
- Implicit usage of attributes (metadata) in the absence of schema is important:
  - "price" not "cost".
- Apache CouchDB:
  - JSON to store data.
  - JavaScript as its query language using Map/Reduce and HTTP for an API.
- MongoDB is also popular.

**Document store**

```json
{ "id": 1001, "customer_id": 7762, "line-items": [ { "product_id": 6543, "quantity": 8 } }, { "product_id": 7656, "quantity": 10 } ]
```

- Inspired by Lotus notes!
- Documents in Jason.
- Jason or JavaScript Object Notation, is an open standard format that uses human-readable text to transmit data objects consisting of attribute-value pairs. It is used primarily to transmit data between a server and web application, as an alternative to XML.

**Metadata and similarities**

- Although key-value and document store are considered two data models, they have some similarities:
  - Both have keys,
  - Some metadata (e.g., customer_id: 6543 is used for indexing (otherwise, have to do sequential search).
- Both allow complex structures to be grouped and stored as a single structure.
- Sometimes called aggregate-oriented databases.
Column-family Databases

- Inspired by Google’s Big Table
- Different from RDBMS column storage such as Sybase IQ
- Column families are defined
- Columns can be defined dynamically
- Storage is more like an array; one dimension is the row identifier or row key, second is the combination of the column-family and column identifier. The third optional dimension is the timestamp and versions.

Column-family Databases (2)

<table>
<thead>
<tr>
<th>Row key</th>
<th>Name</th>
<th>Billing address</th>
<th>Payment</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>&quot;sharma&quot;</td>
<td>&quot;500 UTA Blvd., 640 ERB&quot;</td>
<td>&quot;procard&quot;</td>
</tr>
<tr>
<td>2005</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Order</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order 101</td>
<td>Value</td>
</tr>
<tr>
<td>Order 102</td>
<td>Value</td>
</tr>
<tr>
<td>Order 103</td>
<td>Value</td>
</tr>
<tr>
<td>Order 104</td>
<td>Value</td>
</tr>
</tbody>
</table>

Column-family Databases (3)

- It is also an aggregate-oriented representation
- Can easily distribute data
- Partitioning and distributing facilitates search
- However, in all of the above
  - Sorting and grouping is not easy
  - Report generation is not straightforward
  - Have to slice and dice data for this!
  - Can be done, but takes more computation and effort!
  - Examples: Apache Hbase, MariaDB, ...

Graph Databases

- Data modes uses nodes, relations, K-V on both
- Representation is different from that of tables
- Representation of graph as a table requires too many joins for traversal
- FlockDB developed and used by twitter is a good example
  - Keeps track of who follows whom
  - Traversal from a start node or a set of nodes
  - Ability to add more nodes and edges
  - Graph operations are optimized for this representation
  - Useful for social network analysis!
An example

Graph Databases (2)

- May have their own query language for specifying queries
- Graph representation is becoming more popular as it is easy to see and understand relationships
- "Retrieve all founders of a company who attended Stanford"
- "Query capabilities allow users to look for nodes, scan neighboring nodes, retrieve edges, and retrieve attribute values. Users can also perform more complex queries."

ACID

- We are familiar with ACID properties
  - Atomicity, consistency, isolation, and durability
  - Recovery at the transaction level
  - All of the above come at a cost!
- However, when you update multiple aggregates, how does NoSQL handle that?

ACID (2)

- However, when you update multiple aggregates, how does NoSQL handle that?
  - If the same aggregate is checked out by two users and they update it and post it back, there is a write-write conflict.
  - If we apply ACID, this whole transaction (read, update, and post) needs to be atomic.
  - This also means that the object is locked for a longer period especially if it is interactive! (that is why databases do not allow interactive transactions!)
  - Solution
    - Wrap each action as a Tx
    - Use time version stamp to resolve conflicts
    - By the way RCS and CVS systems have been doing this for a long time
Consistency

- Logical
  - About the data value in a single copy
- Replication
  - **Sharding** does not cause problems (single copy)
  - **distribution with replication** is a different story
    - Availability, resilience etc.
    - Whether all copies are consistent all the time
  - Choice between consistency and availability in the presence of communication failures and partition
    - This choice can be made by the application (or business) instead of the database vendor!
      - Compensation etc.

CAP

- **CAP**
  - Consistency, availability, and (network) partitioning
    - Consistent after the execution of an operation. All clients see the same state after an update
    - availability means no (or very little) down time
    - Tolerance to partition means the system will still be available after a network partition
  - **CAP theorem**
    - In the presence of network partition, you can be either consistent or available, but not both

- **CAP**
  - While traditional databases make that decision for us (which 2 out of 3), NoSQL databases provide these guarantees as tuning/customizing options. Database vendors must always decide which two to prioritize. The options are:
    - Availability is compromised in favor of consistency and partition-tolerance (CP)
    - Partition-tolerance is forfeited in favor of consistency and availability (CA) – mainly for non-distributed systems!
    - Consistency is compromised but systems are always available and can work when parts are partitioned (AP)
BASE

- Basically Available, Soft-state, Eventual consistency
  - Trades off strong consistency guarantees for availability and partition tolerance
  - Promotes availability over consistency by supporting eventual consistency
- BASE is a consistency model used in distributed systems.
  - NoSQL databases promote BASE

Summary

- Vertical scaling – adding more juice to existing boxes
- Horizontal scaling – adding more boxes!
  - Involves data partitioning as well
- Databases are good at vertical scaling, not so good at horizontal scaling
- SQL or NoSQL?
  - If you need horizontal scaling, NoSQL is a better alternative!
  - If your application has no schema or difficult to define a schema (e.g., web log processing, hierarchical documents)
  - If your applications does not need ACID as provided by a RDBMS
  - If you have a graph applications and representation using RDBMS requires too many joins for computation
- Social network applications

Summary (2)

- Good News
  - We have more choices than before
  - You can choose the right model and computation needed to match your application
  - You can get scaling and quick setup
- Bad news
  - You have to write more code
  - NoSQL is still immature
  - Architects have more difficulty in determining and defining what is best for their application
  - Tuning needs to be done on a case-by-case basis
    - Not many available tools
  - We will face Multi-DBMS problems!

Summary (3)

- An example of using different DB technologies for different requirements
  - User sessions (Redis, key-value)
  - Financial data (RDBMS)
  - Shopping Cart (Risk, key-value)
  - Recommendations (Neo4j, graph)
  - Reporting (RDBMS)
  - Product catalog (MongoDB, document store)
- Truly, the amount of non-rdbms data is growing exponentially as compared to rdbms data
- Applications/web services are used for accessing data unlike earlier way of accessing directly
- Applications can mask business-specific logic instead DBMSs
Summary (4)

➢ OldSQL (pronounces old school)
  ▪ Legacy RDBMSs with scalability and performance issues

➢ NoSQL
  ▪ Give up SQL and ACID for performance and scalability

➢ NewSQL
  ▪ Preserve SQL and ACID
  ▪ Get performance from new architecture
  ▪ Being proposed by old schoolers (from DBMS, e.g., Michael Stonebraker, ...)
    - Doubt whether this will succeed!

Thank You !!!