Scalable SQL and NoSQL Data Stores

Rick Cattell

Tulika Thakur
June 1, 2016

University of Waterloo
Table of contents

1. Summary of Paper

2. Critique

3. Competition and Future Work

4. Conclusion
Summary of Paper
Need of Scalability

Some stats:

- **1.65 billion**: Facebook Monthly active users
- **25PB**: size of Internet archive
- **10 million**: strands of DNA Microsoft is buying to store data
- **300TB**: open data from CERN
- **100 billion**: words translated by Google per day
- **8,796**: photos are shared on snapchat every second
- **48,611**: photos posted on Instagram per minute
- **300 hours**: of video uploaded on YouTube per minute

The data production will be **44 TIMES GREATER** in 2020 than it was in 2009.
Thousands or Millions of users doing updates as well as reads!!!
Dimensions of Scalability

- **Horizontal scaling:**
  - increases transaction throughput by splitting the database and associated load over many networked servers. No disk or RAM is shared. (sharding)
  - uses replication for failure recovery.

- **Vertical scaling:**
  - supports higher transaction throughput by more effectively using a single server with a multi-core CPU and lots of memory.
  - in-memory storage, disks used for logging and backups.
  - optimized transactions by avoiding locking and using MVCC
SQL Characteristics

- Data stored in rows and columns - predefined schema.
- Data Manipulation using insert, update, delete.
- Data Aggregation using Group By.
- Compound queries using Joins.
- Functions and Procedures.
- ACID properties (Atomic, Consistent, Isolated, Durable).
NoSQL Characteristics

- No fixed schema.
- Open-source.
- Designed to run in distributed environment.
- Horizontally scalable, supports large amount of data.
- Functions and Procedures.
- BASE properties (BAsically Available, Soft State, Eventually consistent).
- CAP theorem (A distributed system can only have two out of these three properties: consistency, availability, partial-tolerance).
Visual Guide to NoSQL Systems

**Availability:** Each client can always read and write.

**Data Models**
- Relational (comparison)
  - Key-Value
  - Column-Oriented/Tabular
  - Document-Oriented

**CA**
- RDBMSs (MySQL, Postgres, etc)
- Aster Data
- Greenplum
- Vertica

**AP**
- Dynamo
- Voldemort
- Tokyo Cabinet
- KAI
- Cassandra
- SimpleDB
- CouchDB
- Riak

**Pick Two**

**Consistency:** All clients always have the same view of the data.

**CP**
- BigTable
- Hypertable
- Hbase
- MongoDB
- Terrastore
- Scalaris
- Berkeley DB
- MemcacheDB
- Redis

**Partition Tolerance:** The system works well despite physical network partitions.
Data Store Categories

- Key-Value Stores
- Document Stores
- Extensible Record Stores
- Relational Databases
Key-Value Stores

- Hash table for keys.
- Values stored with keys.
- Provides persistence mechanism, replication, versioning, locking, transactions, sorting.
- API: inserts, deletes, index lookups.
- No secondary indices or keys.
<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Address&quot;</td>
<td>“PH05, 222, Albert Street, Waterloo, Canada, N2L3J6”</td>
</tr>
</tbody>
</table>
### Key-Value Stores

**Table 1:** Comparison of some key-value stores

<table>
<thead>
<tr>
<th>DataStore</th>
<th>Distribution</th>
<th>Replication</th>
<th>Concurrency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voldemort</td>
<td>Consistent Hashing</td>
<td>Asynchronous</td>
<td>MVCC</td>
</tr>
<tr>
<td>Riak</td>
<td>Map/Reduce</td>
<td>Asynchronous</td>
<td>MVCC</td>
</tr>
<tr>
<td>Redis</td>
<td>Client controlled</td>
<td>Asynchronous</td>
<td>Locks</td>
</tr>
<tr>
<td>Scalaris</td>
<td>Uses key ranges</td>
<td>Synchronous</td>
<td>Locks</td>
</tr>
<tr>
<td>Memchached</td>
<td>Two phase Hashing</td>
<td>Synchronous</td>
<td>Locks</td>
</tr>
</tbody>
</table>
Document Stores

- Support more complex data: pointerless objects, i.e., documents.
- Secondary indexes, multiple types of documents (objects) per database, nested documents and lists.
- Automatic sharding (scale writes), no explicit locks, weaker concurrency (eventual for scaling reads) and atomicity properties.
- API: select, delete, getAttributes, putAttributes on documents.
- Queries are distributed over multiple nodes using Map-reduce.
Map-Reduce

The MapReduce algorithm contains two important tasks, namely Map and Reduce.

- **Map**: converts data into another set of data, where individual elements are broken down into tuples (key/value pairs). Multiple mapping functions are executed simultaneously, each processing a distinct partition of data.
- **Reduce**: takes the output from a map and combines those data tuples into a smaller set of tuples. A number of reducing functions run concurrently to produce a result set of key/value pairs which form the output of the job.
MapReduce Job – Logical View

Image from - http://mm-tom.s3.amazonaws.com/blog/MapReduce.png
<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
</table>
## Table 2: Comparison of some key-value stores

<table>
<thead>
<tr>
<th>DataStore</th>
<th>Distribution</th>
<th>Replication</th>
<th>Concurrency</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleDB</td>
<td>Domains</td>
<td>Asynchronous</td>
<td>None</td>
</tr>
<tr>
<td>CouchDB</td>
<td>Collections</td>
<td>Asynchronous</td>
<td>MVCC</td>
</tr>
<tr>
<td>MongoDB</td>
<td>Collections</td>
<td>Asynchronous</td>
<td>Locks</td>
</tr>
<tr>
<td>Terrastore</td>
<td>Buckets</td>
<td>Synchronous</td>
<td>Locks</td>
</tr>
</tbody>
</table>
Extensible Record Stores

- Basic data Model is rows and columns
- Supports both Horizontal and Vertical partitioning.
- Motivated by Google’s BigTable.
### Extensible Record Stores

<table>
<thead>
<tr>
<th>Column</th>
<th>Value</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apartment</td>
<td>PH05</td>
<td>1374546754299000</td>
</tr>
<tr>
<td>HouseNumber</td>
<td>222</td>
<td>1374546754299000</td>
</tr>
<tr>
<td>Street</td>
<td>Albert Street</td>
<td>1374546754299000</td>
</tr>
<tr>
<td>Country</td>
<td>Canada</td>
<td>1374546754299000</td>
</tr>
<tr>
<td>City</td>
<td>Waterloo</td>
<td>1374546754299000</td>
</tr>
<tr>
<td>PostalCode</td>
<td>N2L3J6</td>
<td>1374546754299000</td>
</tr>
</tbody>
</table>
### Extensible Record Stores

<table>
<thead>
<tr>
<th>DataStore</th>
<th>Distribution</th>
<th>Replication</th>
<th>Concurrency</th>
</tr>
</thead>
<tbody>
<tr>
<td>HBase</td>
<td>MapReduce</td>
<td>Asynchronous</td>
<td>Locks and Logging</td>
</tr>
<tr>
<td>HyperTable</td>
<td>Key-range</td>
<td>Synchronous</td>
<td>Locks and Logging</td>
</tr>
<tr>
<td>Cassandra</td>
<td>Automatic</td>
<td>Asynchronous</td>
<td>MVCC</td>
</tr>
</tbody>
</table>
Relational Data Stores

Predefined schema, ACID properties, SQL interface. It provides scalability using two provisions-

2. Small-scope Transactions.
Relational Data Stores

**Table 4:** TableName- Address

<table>
<thead>
<tr>
<th>Apartment</th>
<th>HouseNumber</th>
<th>Street</th>
<th>City</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;PH05&quot;</td>
<td>&quot;222&quot;</td>
<td>&quot;Albert Street&quot;</td>
<td>&quot;Waterloo&quot;</td>
<td>&quot;Canada&quot;</td>
</tr>
</tbody>
</table>
# Relational Data Stores

## Table 5: Comparison of some key-value stores

<table>
<thead>
<tr>
<th>DataStore</th>
<th>Distribution</th>
<th>Replication</th>
<th>Concurrency</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL Cluster</td>
<td>Sharding</td>
<td>Bi-directional</td>
<td>ACID</td>
</tr>
<tr>
<td>VoltDB</td>
<td>Sharding</td>
<td>Synchronous</td>
<td>ACID</td>
</tr>
<tr>
<td>Clustrix</td>
<td>Automatic</td>
<td>Synchronous</td>
<td>ACID</td>
</tr>
<tr>
<td>ScaleDB</td>
<td>sharding(disks shared)</td>
<td>Synchronous</td>
<td>ACID</td>
</tr>
<tr>
<td>ScaleBase</td>
<td>Sharding</td>
<td>Asynchronous</td>
<td>ACID</td>
</tr>
<tr>
<td>NimbusDB</td>
<td>Sharding</td>
<td>Synchronous</td>
<td>ACID</td>
</tr>
</tbody>
</table>
Use the Right tool for the Right job!!!
Use Cases- Key-value store

Simple application, with simple objects which need to be looked up on the basis of single attribute. Very few updates, more reads.

SELECT name, pic, profileURL from user where userId = 'Tulika';
Similar to key-value stores, but when look-up needs to be done on multiple attributes, objects are complex too.

Eventually consistent, limited atomicity and isolation.

SELECT licenseNumber from DMV where userId = 'Tulika';
Use Cases- Extensible Record store

Similar to document stores- look-up needs to be done on multiple attributes.

Stronger concurrency(locks), with added complexity.

Systems which require both horizontal and vertical partitioning of data.

Example- shopping sites.
Simplifies data definition and relations.

ACID properties maintained by DBMS.

Data being updated from various locations.
### Table 6: Comparison of SQL, NoSQL, NewSQL

<table>
<thead>
<tr>
<th></th>
<th>SQL</th>
<th>NoSQL</th>
<th>NewSQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>No scalability</td>
<td></td>
<td>High Availability</td>
<td>Weak availability</td>
</tr>
<tr>
<td>Fixed schema</td>
<td></td>
<td>Flexible</td>
<td>Fixed schema</td>
</tr>
<tr>
<td>Structures Data</td>
<td></td>
<td>Non-Structures Data</td>
<td>Structures Data</td>
</tr>
<tr>
<td>Disk Based</td>
<td></td>
<td>In-memory or Disk Based</td>
<td>In-memory or Disk</td>
</tr>
<tr>
<td>ACID</td>
<td></td>
<td>BASE</td>
<td>ACID</td>
</tr>
<tr>
<td>Complex consistency</td>
<td></td>
<td>Eventual Consistency</td>
<td>Strong Consistency</td>
</tr>
</tbody>
</table>
Critique
• No Benchmark results shared. Benchmark results comparing NoSQL - NewSQL missing!
• NoSQL is still immature, should be used only where applicable. The security capabilities of NoSQL solutions are limited.
• Cannot abandon SQL completely.
• NoSQL does not provide good consistency. Almost all of the NoSQL datastores discussed here follow AP. Is high consistency not important?
• NewSQL- Good choice for existing SQL applications and emerging digital applications that have data dependencies across multiple rows or documents.
• Although an analysis platform called Pig provides some basic analytical functionalities for NoSQL data stores, it is not yet as powerful as the BI tools available for RDBMSs.
• Polygot Persistence - Facebook uses HBase for mesaging service, Cassandra for other services.
Balanced Read/Write Mix

Operations/sec

Nodes

Cassandra  Couchbase  HBase  MongoDB

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Cassandra</th>
<th>Couchbase</th>
<th>HBase</th>
<th>MongoDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13,929.58</td>
<td>1,554.14</td>
<td>527.47</td>
<td>1,278.81</td>
</tr>
<tr>
<td>2</td>
<td>28,078.26</td>
<td>2,985.28</td>
<td>1,503.09</td>
<td>1,441.32</td>
</tr>
<tr>
<td>4</td>
<td>51,111.84</td>
<td>3,755.28</td>
<td>4,175.8</td>
<td>1,801.06</td>
</tr>
<tr>
<td>8</td>
<td>95,005.27</td>
<td>10,138.80</td>
<td>7,725.94</td>
<td>2,195.92</td>
</tr>
<tr>
<td>16</td>
<td>172,668.48</td>
<td>11,761.31</td>
<td>16,381.78</td>
<td>1,230.96</td>
</tr>
<tr>
<td>32</td>
<td>302,181.72</td>
<td>21,375.02</td>
<td>20,177.71</td>
<td>2,335.14</td>
</tr>
</tbody>
</table>
Competition and Future Work
Splice Machine
Splice Machine is a Hadoop RDBMS built on top of HBase/Hadoop.

MySQL Cluster
It provides shared-nothing clustering and auto-sharding for the MySQL database management system.

Benchmark
Little has been done to compare the performance of different NoSQL solutions under different processing loads. Yahoo Cloud Serving Benchmark (YCSB) is the only one.

Interface
RDBMS have common interface, should be a common interface for accessing key-value, document, and column-family data stores.
Conclusion
• NoSQL not designed to replace SQL databases.
• RDBMS - good in handling OTLP transactions. (performance, ACID compliant)
• Cannot migrate a 25-node MongoDB cluster to Oracle databases.
• Technology does not give you performance, Architecture does!!
References I

Questions?