Advanced Data Management (CSCI 640/490)

Scalable Dataframes

Dr. David Koop
Recent History in Databases

- Early 2000s: Commercial DBs dominated, Open-source DBs missing features
- Mid 2000s: MySQL adopted by web companies
- Late 2000s: NoSQL does scale horizontally out of the box
- Early 2010s: New DBMSs that can scale across multiple machines natively and provide ACID guarantees
NewSQL Definitions

• Stonebraker's Definition:
  - SQL as the primary interface
  - ACID support for transactions
  - Non-locking concurrency control
  - High per-node performance
  - Parallel, **shared-nothing** architecture (what about shared-disk?)

• Wikipedia (Pavlo): A class of modern relational DBMSs that provide the same scalable performance of NoSQL systems for OLTP workloads while still maintaining the ACID guarantees of a traditional DBMS.
NewSQL Positioning

<table>
<thead>
<tr>
<th>SCALABILITY</th>
<th>LOW (One Node)</th>
<th>HIGH (Many Nodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOSQL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEWSQL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRADITIONAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WEAK (None/Limited)</td>
<td></td>
<td>STRONG (ACID)</td>
</tr>
</tbody>
</table>
Three Types of NewSQL Systems

• New Architectures
  - New codebase without architectural baggage of legacy systems
  - Examples: VoltDB, Spanner, Clustrix

• Transparent Sharding Middleware:
  - Transparent data sharding & query redirecting over cluster of single-node DBMSs
  - Examples: citusdata, ScaleArc (usually support MySQL/postgres wire)

• Database-as-a-Service:
  - Distributed architecture designed specifically for cloud-native deployment
  - Examples: xeround, GenieDB, FathomDB (usually based on MySQL)
What went wrong?

- Almost every NewSQL company from the last decade has closed, sold for scraps, or pivoted to other markets
- Why?
  - Selling an OLTP Database System is hard
  - Startup cost of a relational system is harder than NoSQL
  - Existing DBMS Systems (MySQL, postgresql) are Good
  - Cloud Disruption
    - Can't sell on-premises
    - Can't complete on cost with cloud vendors
  - Lack of Open Source
Conclusions

• NewSQL is dead
• Academic: the NewSQL movement was a success
• Business: a failure for those who embraced the NewSQL mantle
• Next?
  - You still need humans to design, configure, and optimize logical/physical aspects of a database
  - Humans are expensive
  - Automation is the future.

[A. Pavlo]
Spanner Overview

- Focus on scaling databases focused on OLTP (not OLAP)
- Since OLTP, focus is on sharding **rows**
- Tries to satisfy CAP (which is impossible per CAP Theorem) by not worrying about 100% availability
- External consistency using multi-version concurrency control through timestamps
- ACID is important
- Structured: universe with zones with zone masters and then spans with span masters
- SQL-like (updates allow SQL to be used with Spanner)
Spanner and the CAP Theorem

- Which type of system is Spanner?
  - C: consistency, which implies a single value for shared data
  - A: 100% availability, for both reads and updates
  - P: tolerance to network partitions

- Which two?
  - CA: close, but not totally available
  - So actually CP
External Consistency

• Traditional DB solution: **two-phase locking**—no writes while client reads

• "The system behaves as if all transactions were executed sequentially, even though Spanner actually runs them across multiple servers (and possibly in multiple datacenters) for higher performance and availability" [Google]

• Semantically indistinguishable from a single-machine database

• Uses multi-version concurrency control (MVCC) using **timestamps**

• Spanner uses **TrueTime** to generate monotonically increasing timestamps across all nodes of the system
Google Cloud Spanner

- https://cloud.google.com/spanner/
- Features:
  - Global Scale: thousands of nodes across regions / data centers
  - Fully Managed: replication and maintenance are automatic
  - Transactional Consistency: global transaction consistency
  - Relational Support: Schemas, ACID Transactions, SQL Queries
  - Security
  - Highly Available
<table>
<thead>
<tr>
<th>Frequency</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.30% INSERT INTO <code>terms</code> (<code>term</code>, <code>rank</code>, <code>set_id</code>, <code>last_modified</code>) VALUES (?,?,?,?),(?,?,?,?)</td>
</tr>
<tr>
<td>2</td>
<td>0.25% INSERT INTO <code>terms</code> (<code>term</code>, <code>rank</code>, <code>set_id</code>, <code>last_modified</code>, <code>definition</code>) VALUES (?,?,?,?),(?,?,?,?),(?,?,?,?)(?,?,?,?)</td>
</tr>
<tr>
<td>3</td>
<td>4.22% INSERT INTO <code>terms</code> (<code>term</code>, <code>rank</code>, <code>set_id</code>, <code>last_modified</code>) VALUES (?,?,?,?)</td>
</tr>
<tr>
<td>4</td>
<td>1.88% INSERT INTO <code>terms</code> (<code>term</code>, <code>rank</code>, <code>set_id</code>, <code>last_modified</code>, <code>definition</code>) VALUES (?,?,?,?)</td>
</tr>
<tr>
<td>5</td>
<td>3.28% SELECT * FROM <code>terms</code> WHERE (<code>is_deleted</code> = 0) AND (<code>set_id</code> IN (??)) AND (<code>rank</code> IN (0,1,2,3)) AND (<code>term</code> != ”)</td>
</tr>
<tr>
<td>6</td>
<td>14.13% SELECT <code>set_id</code>, COUNT(*) FROM <code>terms</code> WHERE (<code>is_deleted</code> = 0) AND (<code>set_id</code> = ?) GROUP BY <code>set_id</code></td>
</tr>
<tr>
<td>7</td>
<td>12.56% SELECT * FROM <code>terms</code> WHERE (<code>id</code> = ?)</td>
</tr>
<tr>
<td>8</td>
<td>0.49% SELECT * FROM <code>terms</code> WHERE (<code>id</code> IN (??)) AND (<code>set_id</code> IN (??))</td>
</tr>
<tr>
<td>9</td>
<td>4.11% SELECT <code>id</code>, <code>set_id</code> FROM <code>terms</code> WHERE (<code>set_id</code> = ?) LIMIT 20000</td>
</tr>
<tr>
<td>10</td>
<td>0.43% SELECT <code>id</code>, <code>set_id</code> FROM <code>terms</code> WHERE (<code>set_id</code> IN (??)) LIMIT 20000</td>
</tr>
<tr>
<td>11</td>
<td>0.59% SELECT * FROM <code>terms</code> WHERE (<code>id</code> IN (??))</td>
</tr>
<tr>
<td>12</td>
<td>36.76% SELECT * FROM <code>terms</code> WHERE (<code>set_id</code> = ?)</td>
</tr>
<tr>
<td>13</td>
<td>0.61% SELECT * FROM <code>terms</code> WHERE (<code>set_id</code> IN (??))</td>
</tr>
<tr>
<td>14</td>
<td>6.10% UPDATE <code>terms</code> SET <code>definition</code>=?, <code>last_modified</code>=? WHERE <code>id</code>=? AND <code>set_id</code>=?</td>
</tr>
<tr>
<td>15</td>
<td>0.33% UPDATE <code>terms</code> SET <code>is_deleted</code>=?, <code>last_modified</code>=? WHERE <code>id</code> IN (??) AND <code>set_id</code>=?</td>
</tr>
<tr>
<td>16</td>
<td>12.56% UPDATE <code>terms</code> SET <code>rank</code>=?, <code>last_modified</code>=? WHERE <code>id</code>=? AND <code>set_id</code>=?</td>
</tr>
<tr>
<td>17</td>
<td>1.06% UPDATE <code>terms</code> SET <code>word</code>=?, <code>last_modified</code>=? WHERE <code>id</code>=? AND <code>set_id</code>=?</td>
</tr>
<tr>
<td>18</td>
<td>0.32% UPDATE <code>terms</code> SET <code>definition</code>=?, <code>word</code>=?, <code>last_modified</code>=? WHERE <code>id</code>=? AND <code>set_id</code>=?</td>
</tr>
</tbody>
</table>

[P. Bakkum and D. Cepeda, 2017]
Latency: Spanner vs. MySQL

Latency at 3,000 Queries per Second

[P. Bakkum and D. Cepeda, 2017]
Latency: Spanner vs. MySQL

Latency at 9,000 Queries per Second

- **Spanner**
- **MySQL**

Median Latency (ms)

Query

[P. Bakkum and D. Cepeda, 2017]
Throughput: Spanner vs. MySQL

[Graph showing median latency as throughput increases]

- MySQL (median)
- Spanner 9 nodes (median)
- Spanner 15 nodes (median)
- Spanner 30 nodes (median)

[P. Bakkum and D. Cepeda, 2017]
Max Throughput vs. Nodes

[P. Bakkum and D. Cepeda, 2017]
Spanner: Latency vs. Nodes

Latency at 3000 QPS vs Nodes

Latencies @ 3000 QPS

Nodes

[P. Bakkum and D. Cepeda, 2017]
Assignment 4

• Work on Data Integration and Data Fusion
• Integrate travel datasets from different institutions (UN World Tourism Office, World Bank, OECD)
  - Integrate information with population
• Record Matching:
  - Which countries are the same?
• Data Fusion:
  - The receipts/expenditures
  - Country names
Scalable Dataframes
History of Dataframes

- R, open-source alternative to S, developed in 2000 (with dataframes)
- Pandas, 2009
- Spark, 2010 (resilient distributed dataset [RDD], Dataset API)
Formalizing Dataframes

- Combines parts of matrices, databases, and spreadsheets
- Ordered rows (unlike databases)
- Types can be inferred at runtime, not the same across all columns
- Lots of "intuitive" functions (600+)
Differences between Databases & Dataframes

Convenience

Entire query at once

Flexible

Strict schema

Versatility

SFW or bust

Incremental + inspection

Mixed types, R/C and data/metadata equiv.

600+ functions

[D. Petersohn, 2022]
Scaling Dataframes

• Solutions:
  - Spark
  - Dask
  - Polars
  - Vaex
  - Modin
Scaling up your pandas workflows with Modin

D. Petersohn
Beyond pandas: The great Python dataframe showdown

J. L. C. Rodríguez