Carnegie Mellon Univ.
Dept. of Computer Science
15-415/615 - DB Applications

C. Faloutsos – A. Pavlo
Lecture#25: OldSQL vs. NoSQL vs. NewSQL

OLTP vs. OLAP

• On-line Transaction Processing:
  – Short-lived txns.
  – Small footprint.
  – Repetitive operations.

• On-line Analytical Processing:
  – Long running queries.
  – Complex joins.
  – Exploratory queries.

The Boring Days (1990s)

• Microsoft forks Windows version of Sybase code and creates SQL Server.
• MySQL released as a replacement for mSQL.
• Postgres gets SQL support.
• Illustra bought by Informix.

Internet Boom (2000s)

• New Internet start-ups hit the performance and cost limits of “elephant” DBMSs.
• Early companies used custom middleware to shard databases across multiple DBMSs.
• Google was a pioneer in developing non-relational DBMS architectures.
MapReduce

- Simplified parallel computing paradigm for large-scale data analysis.
- Originally proposed by Google in 2004.
- Hadoop is the current leading open-source implementation.

MapReduce Example

Calculate total order amount per day after Jan 1st.

```
MAP(key, value) {
   if (key >= "2009-01-01") {
      output(key, value);
   }
}

REDUCE(key, values) {
   sum = 0;
   while (values.hasNext()) {
      sum += values.next();
   }
   output(key, sum);
}
```

What MapReduce Does Right

- Since all intermediate results are written to HDFS, if one node crashes the entire query does not need to be restarted.
- Easy to load data and start running queries.
- Great for semi-structured data sets.

What MapReduce Did Wrong

- Have to parse/cast values every time:
  - Multi-attribute values handled by user code.
  - If data format changes, code must change.
- Expensive execution:
  - Have to send data to executors.
  - A simple join requires multiple MR jobs.
Join Example

- Find sourceIP that generated most adRevenue along with its average pageRank.

Join Example – SQL

```sql
SELECT INTO Temp
    sourceIP,
    AVG(pageRank) AS avgPageRank,
    SUM(adRevenue) AS totalRevenue
FROM Rankings AS R, UserVisits AS UV
WHERE R.pageURL = UV.destURL
  AND UV.visitDate BETWEEN "2000-01-15" AND "2000-01-22"
GROUP BY UV.sourceIP;

SELECT sourceIP, totalRevenue, avgPageRank
  FROM Temp ORDER BY totalRevenue DESC LIMIT 1;
```

Join Example – MapReduce

Phase 1: Filter
- Map: Emit all records for Rankings.
- Filter UserVisits data.
- Reduce: Compute cross product.

Phase 2: Aggregation
- Map: Emit all tuples (i.e., passthrough)
- Reduce: Compute avg pageRank for each sourceIP.

Phase 3: Search
- Map: Emit all tuples (i.e., passthrough)
- Reduce: Scan entire input and emit the record with greatest adRevenue sum.

Join Example – Results

- Find sourceIP that generated most adRevenue along with its average pageRank.

<table>
<thead>
<tr>
<th></th>
<th>25 nodes</th>
<th>50 nodes</th>
<th>100 nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop</td>
<td>32.0</td>
<td>35.4</td>
<td>35.0</td>
</tr>
<tr>
<td>Vertica</td>
<td>29.2</td>
<td>29.4</td>
<td>33.9</td>
</tr>
<tr>
<td>DBMS-X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Distributed Joins Are Hard

- Assume tables are horizontally partitioned:
  - Table1 Partition Key → table1.key
  - Table2 Partition Key → table2.key

- Q: How to execute?
- Naïve solution is to send all partitions to a single node and compute join.

Semi-Joins

- First distribute the join attributes between nodes and then recreate the full tuples in the final output.
  - Send just enough data from each table to compute which rows to include in output.

- Lots of choices make this problem hard:
  - What to materialize?
  - Which table to send?

MapReduce in 2015

- Database Connectors.
- SQL/Declarative Query Support.
- Table Schemas.
- Column-oriented storage.

Column Stores

- Store tables as sections of columns of data rather than as rows of data.
- Only scan the columns that a query needs.
- Allows for amazing compression ratios.
Column Stores

Query 1:
```sql
SELECT sex, AVG(GPA) FROM student GROUP BY sex
```

Row-oriented Storage

- Rows are stored as a flat list
- Each row contains a single record

Column-oriented Storage

- Columns are stored separately
- Each column is stored as a separate list

Advantages of Column Stores:
- Delay materializing a record for as long as possible inside of the DBMS.
- Pre-sorting can improve compression:
  - Example: Run-length Encoding
- Inserts/Updates/Deletes are harder…

Column Store Systems

- Many column-store DBMSs
  - Examples: Vertica, Sybase IQ, MonetDB
- Hadoop storage library:
  - Example: Parquet, RCFile
The Rise of NoSQL (2000s)

• Developers spend time writing middleware rather than working on core applications.
• Google created a distributed DBMS called BigTable in 2004:
  – It used a GET/PUT API instead of SQL.
  – No support for txns.
• Newer systems have been created that follow BigTable’s anti-relational spirit.

NoSQL Systems

Key/Value

Column-Family

Documents

Documents

Column-Family

Key/Value

NewSQL (2010s)

• Next generation of relational DBMSs that can scale like a NoSQL system but without giving up SQL or txns.

NoSQL Drawbacks

• Developers write code to handle eventually consistent data, lack of transactions, and joins.
• Not all applications can give up strong transactional semantics.
Aslett White Paper

[Systems that] deliver the scalability and flexibility promised by NoSQL while retaining the support for SQL queries and/or ACID, or to improve performance for appropriate workloads.

Wikipedia Article

A class of modern relational database systems that provide the same scalable performance of NoSQL systems for OLTP workloads while still maintaining the ACID guarantees of a traditional database system.

NewSQL Systems

- New Design
- MySQL Engines
- Middleware

Faloutsos/Pavlo

NewSQL Implementations

- Distributed Concurrency Control
- Main Memory Storage
- Hybrid Architectures
  - Support OLTP and OLAP in single DBMS.
- Query Code Compilation

Faloutsos/Pavlo
Summary

SCALABILITY

HIGH
(Many Nodes)

LOW
(One Node)

WEAK
(None/Limited)

GUARANTEES

STRONG
(ACID)

NO SQL
NEWSQL

TRADITIONAL