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.

```
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

```
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.

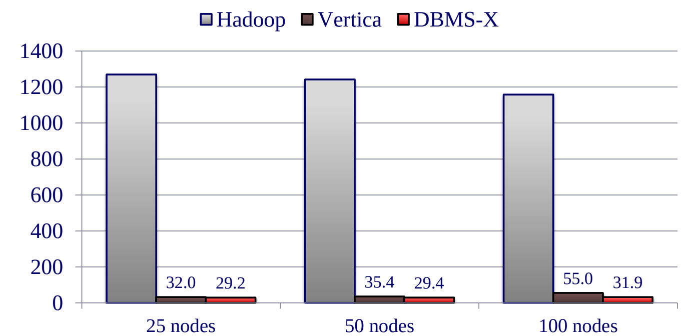
Phase 3:
Search

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.



Distributed Joins Are Hard

```
SELECT * FROM table1, table2
 WHERE table1.val = table2.val
```

- 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.

HIVE

Hortonworks STINGER

cloudera IMPALA

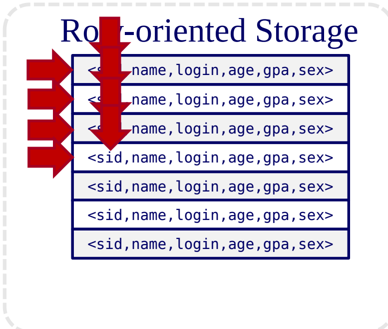
Column Stores

- 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

```
SELECT sex, AVG(GPA) FROM student
  GROUP BY sex
```

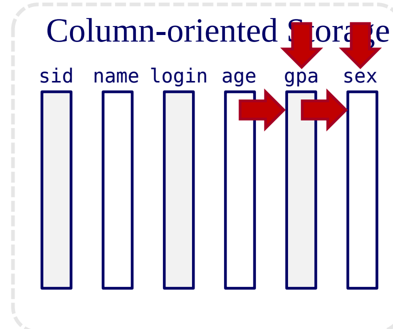
sid	name	login	age	gpa	sex
1001	Faloutsos	christos@cs	45	4.0	M
1002	Bieber	jbieber@cs	21	3.9	M
1003	Tupac	shakur@cs	26	3.5	M
1004	Ke$sha	kesha@cs	22	4.0	F
1005	LadyGaGa	gaga@cs	24	3.5	F
1006	Obama	obama@cs	50	3.7	M



Column Stores

```
SELECT sex, AVG(GPA) FROM student
  GROUP BY sex
```

sid	name	login	age	gpa	sex
1001	Faloutsos	christos@cs	45	4.0	M
1002	Bieber	jbieber@cs	21	3.9	M
1003	Tupac	shakur@cs	26	3.5	M
1004	Ke$sha	kesha@cs	22	4.0	F
1005	LadyGaGa	gaga@cs	24	3.5	F
1006	Obama	obama@cs	50	3.7	M



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

redis
riak
FOUNDATIONDB
membase
amazon DynamoDB

Column-Family

cassandra
APACHE HBASE
HYPERTABLE

Documents

mongoDB
COUCHBASE
CouchDB relax
RethinkDB

NoSQL Drawbacks

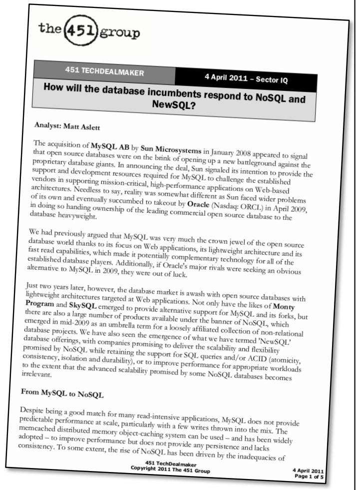
- Developers write code to handle eventually consistent data, lack of transactions, and joins.
- Not all applications can give up strong transactional semantics.

NewSQL (2010s)

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

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.



the 451 group

451 TECHDEALMAKER | 4 April 2011 – Sector IQ

How will the database incumbents respond to NoSQL and NewSQL?

Analyst: Matt Aslett

The acquisition of **MySQL AB** by **Sun Microsystems** in January 2008 appeared to signal that open source databases were on the brink of opening up a new battleground against the proprietary database giants. In announcing the deal, Sun signaled its intention to provide the support and development resources required for MySQL to challenge the established vendors in supporting mission-critical, high-performance applications on Web-based architectures. Needless to say, reality was somewhat different as Sun faced wider problems of its own and eventually succumbed to takeout by **Oracle** (Nasdaq: ORCL) in April 2009, in doing so handing ownership of the leading commercial open source database to the database heavyweight.

We had previously argued that MySQL was very much the crown jewel of the open source database world thanks to its focus on Web applications, its lightweight architecture and its fast read capabilities, which made it potentially complementary technology for all of the established database players. Additionally, if Oracle's major rivals were seeking an obvious alternative to MySQL in 2009, they were out of luck.

Just two years later, however, the database market is awash with open source databases with lightweight architectures targeted at Web applications. Not only have the likes of **Monty Program** and **SkySQL** emerged to provide alternative support for MySQL and its forks, but there are also a large number of products available under the banner of NoSQL, which emerged in mid-2009 as an umbrella term for a loosely affiliated collection of non-relational database projects. We have also seen the emergence of what we have termed 'NewSQL' database offerings, with companies promising to deliver the scalability and flexibility promised by NoSQL while retaining the support for SQL queries and/or ACID (atomicity, consistency, isolation and durability), or to improve performance for appropriate workloads to the extent that the advanced scalability promised by some NoSQL databases becomes irrelevant.

From MySQL to NoSQL

Despite being a good match for many read-intensive applications, MySQL does not provide predictable performance at scale, particularly with a few writes thrown into the mix. The memcached distributed memory object-caching system can be used – and has been widely adopted – to improve performance but does not provide any persistence and lacks consistency. To some extent, the rise of NoSQL has been driven by the inadequacies of

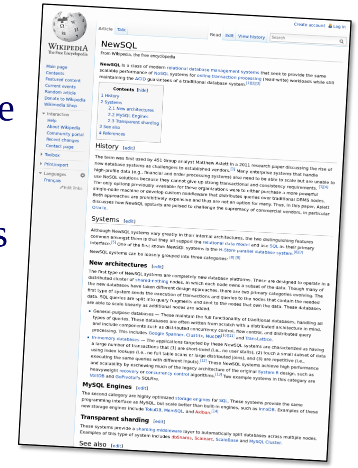
451 TechDealmaker
Copyright 2011 The 451 Group

4 April 2011
Page 1 of 5

Matt Aslett – 451 Group (April 4th, 2011)
https://www.451research.com/report-short?entityId=66963

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

From Wikipedia, the free encyclopedia

NewSQL is a class of modern relational database management systems that seek to provide the same scalable performance of NoSQL systems for online transaction processing (read-write) workloads while still maintaining the ACID guarantees of a traditional database system.[1][2][3]

Contents [hide]
1 History
2 Systems
2.1 New architectures
2.2 MySQL Engines
2.3 Transparent sharding
3 See also
4 References

History [edit]

The term was first used by 451 Group analyst Matthew Aslett in a 2011 research paper discussing the rise of new database systems as challengers to established vendors.[1] Many enterprise systems that handle high-profile data (e.g., financial and order processing systems) also need to be able to scale but are unable to use NoSQL solutions because they cannot give up strong transactional and consistency requirements.[1][4] The only options previously available for these organizations were to either purchase a more powerful single-node machine or develop custom middleware that distributes queries over traditional DBMS nodes. Both approaches are prohibitively expensive and thus are not an option for many. Thus, in this paper, Aslett discusses how NewSQL upstarts are poised to challenge the supremacy of commercial vendors, in particular Oracle.

Systems [edit]

Although NewSQL systems vary greatly in their internal architectures, the two distinguishing features common amongst them is that they all support the relational data model and use SQL as their primary interface.[5] One of the first known NewSQL systems is the H-Store parallel database system.[6][7]

NewSQL systems can be loosely grouped into three categories: [8] [9]

New architectures [edit]

The first type of NewSQL systems are completely new database platforms. These are designed to operate in a distributed cluster of shared-nothing nodes, in which each node owns a subset of the data. Though many of the new databases have taken different design approaches, there are two primary categories evolving. The first type of system sends the execution of transactions and queries to the nodes that contain the needed data. SQL queries are split into query fragments and sent to the nodes that own the data. These databases are able to scale linearly as additional nodes are added.

- General-purpose databases — These maintain the full functionality of traditional databases, handling all types of queries. These databases are often written from scratch with a distributed architecture in mind, and include components such as distributed concurrency control, flow control, and distributed query processing. This includes Google Spanner, Clustrix, NuoDB[10][11] and TransLattice.
- In-memory databases — The applications targeted by these NewSQL systems are characterized as having a large number of transactions that (1) are short-lived (i.e., no user stalls), (2) touch a small subset of data using index lookups (i.e., no full table scans or large distributed joins), and (3) are repetitive (i.e., executing the same queries with different inputs).[12] These NewSQL systems achieve high performance and scalability by eschewing much of the legacy architecture of the original System R design, such as heavyweight recovery or concurrency control algorithms.[13] Two example systems in this category are VoltDB and GoPivotal's SQLFire.

MySQL Engines [edit]

The second category are highly optimized storage engines for SQL. These systems provide the same programming interface as MySQL, but scale better than built-in engines, such as InnoDB. Examples of these new storage engines include TokuDB, MemSQL, and Akiban.[14]

Transparent sharding [edit]

These systems provide a sharding middleware layer to automatically split databases across multiple nodes. Examples of this type of system includes dbShards, Scalearc, ScaleBase and MySQL Cluster.

See also [edit]

Wikipedia (April 2015)
http://en.wikipedia.org/wiki/NewSQL

NewSQL Systems

New Design

NUODB
VOLTDB
SAP HANA
FOUNDATIONDB
memsql
Clustrix
Google Spanner

MySQL Engines

deepdb
ScaleDB
Tokutek

Middleware

ScaleBase
ScaleArc
dbShards

NewSQL Implementations

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

Summary


SCALABILITY
HIGH
(Many Nodes)
LOW
(One Node)
NoSQL
NewSQL
TRADITIONAL
WEAK
(None/Limited)
GUARANTEES
STRONG
(ACID)