HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads

Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel J. Abadi, Alexander Rasin and Avi Silberschatz

Presented by Xiaolu Xiong, Jiefeng He
Problem and Motivation

Data is exploding

Database system is moving to cheap, lower-end, commodity hardware

Market is huge
Two common approaches

Parallel DBs vs. MapReduce
Parallel DBs

Work with SQL on relational tables, and efficient

Scalable, but in tens nodes not in hundreds, also with assumptions:
   Node failure is a rare event
   Homogeneous machines

And expensive!!!
MapReduce

Work on unstructured data

Scale to a cluster of thousands

Node failure is expected

Cheap!!!
What if we put them together?

HadoopDB

Performance

Fault Tolerance

Heterogeneous Environment

Flexible Interface
Performance

The primary characteristic of database systems is their parallel processing capabilities. Parallel DBs: decades of optimizations on relational tables.
Fault Tolerance

Google: average of 1.2 failures per job

MapReduce: Nodes failed? idc ...
Heterogeneous Environment

Impossible to get homogeneous across hundreds of nodes

Parallel DBs: slowest node determines job completion

MapReduce: reassigns task of slow nodes
Flexible Interface

Everybody knows SQL

Parallel DBMSs: modeled data and declarative SQL
HadoopDB

Each machine runs a single-node DB (Postgres)
   Push as much as possible down there

Tied together with MapReduce on top as coordinator and communication layer

Queries primarily expressed in SQL
Hadoop ...

Storage: HDFS

MapReduce framework

Interface between them
HadoopDB: Database Connector

Extend Hadoop's inputFormat class

Interface between DB on the nodes and TaskTrackers
HadoopDB: Catalog

Metadata about DBs in Node

Connection parameters

Table locations
HadoopDB: Data Loader

Global Hasher partitions raw data from HDFS

Local Hasher partition into Local FS and smaller chunks
Table: saleDate | revenue
2001/2    2M
2001/3    1.5M
..        ..
2004/3    2M

Partition: Using YEAR(saleDate) as key
HadoopDB: S(QL) - M(apReduce) - S

Extend Hive, Hive already transforms HiveQL (almost SQL) to MapReduce
HadoopDB: SMS

Hive assumes no collocation of tables

All join and Groupby are in reducers
Hive: SELECT YEAR(saleDate), SUM(revenue) FROM sales GROUP BY YEAR(saleDate)
HadoopDB: SMS

SMS modifies Hive to use custom catalog.

All operators work with local keys.
HadoopDB: SM

SMS: SELECT YEAR(saleDate), SUM(revenue) FROM sales GROUP BY YEAR(saleDate)
Evaluating HadoopDB

Hadoop: hand-coded && Hive-coded

HadoopDB: hand-coded && SMS-coded

Vertica: column-store

DBMS-X: (performance estimated)
Grep Task
SELECT * FROM Data WHERE field LIKE `%XYZ%`

Selection Task
SELECT pageURL, pageRank FROM Rankings WHERE pageRank > 10;

Aggregation Task
SELECT SUBSTR(sourceIP, 1, 7), SUM(adRevenue)
FROM UserVisits GROUP BY SUBSTR(sourceIP, 1, 7);
SELECT sourceIP, SUM(adRevenue) FROM UserVisits
GROUP BY sourceIP;

Join Task
SELECT sourceIP, COUNT(pageRank), SUM(pageRank),
SUM(adRevenue) 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;

UDF Aggregation Task
SELECT url, contents FROM Documents;
Data Loading

**Grep:** no preprocessing

**UserVisit:** repartitioned by destinationURL, indexed by visitDate

V,H,H ensure all system scale in contrast to DB-X
Results: Grep & Selection

Figure 5: Grep Task

Figure 6: Selection Task
Results: Large & Small Aggregation

Figure 7: Large Aggregation Task

Figure 8: Small Aggregation Task
Results: Join & UDF Aggregation

Figure 9: Join Task

Figure 10: UDF Aggregation task
Summary thus far

In the absence of failures ro background process: HadoopDB ~ parallel database systems
1. PostgreSQL not column-store
2. DBMS-X overly optimistic by 15%
3. no data compression in PostgreSQL
4. overhead exists: Hadoop vs PostgreSQL

HadoopDB > Hadoop
more loading time --> better performance
Scalbility

Figure 11: Fault tolerance and heterogeneity experiments on 10 nodes
Discussion

Vertica had a larger percentage slowdown, but total query time was less

Vertica is faster in the absence of failures
   Could be duplicated in HadoopDB with fewer nodes
   96 nodes still has rare failure rate

Trend prediction: 1,000-node and 10,000-node clusters will be more common in the future
   Increased probability for failure
   HadoopDB will outperform in these situation
   Can control the tradeoff between fault tolerance and performance in HadoopDB using chunk size
Conclusion

HadoopDB:

Hybrid system
Scales better than parallel databases
Fault tolerant
Approaches the performance of parallel databases
Free and open source
Thank you!

Any Questions?