Seminar on Distributed Information Systems

Dr.-Ing. Sebastian Michel
Dr.-Ing. Martin Theobald

Today's topic:

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

presented by Frederic Raber
Outline

Motivation & Goals

HadoopDB - Architecture

Hadoop Extensions

HadoopDB – Components

Benchmark
Two Philosophies

- Use few expensive high-end machines → hardware failures very rare
Two Philosophies

- Use few expensive high-end machines → hardware failures very rare
- Parallelize queries by replicating & partitioning
Two Philosophies

- Use few expensive high-end machines → hardware failures very rare
- Parallelize queries by replicating & partitioning → very high performance → bad fault-tolerance → bad scalability → high cost for hardware
Example: parallel DBMS
Example: parallel DBMS

- Partition input data or make replicas on several machines
Example: parallel DBMS

- Partition input data or make replicas on several machines
- Work on it in parallel
- Join the Output
Example: parallel DBMS

- Partition input data or make replicas on several machines
- Work on it in parallel
- Join the Output
- Detailed information: see core lecture „Database Systems“ by Jens Dittrich
- A lot of high-performant, commercial implementations (Vertica, DB-X)
Two Philosophies

- Use a lot of cheap hardware
  → HW failures are the rule, not the exception
Two Philosophies

- Use a lot of cheap hardware
  \(\rightarrow\) \textit{HW failures are the rule, not the exception}

- Parallelize queries by transforming into map-reduce tasks
Two Philosophies

• Use a lot of cheap hardware
  → HW failures are the rule, not the exception

• Parallelize queries by transforming into map-reduce tasks
  → good fault-tolerance
  → good scalability
  → low cost for hardware
  → usually bad performance
Example: Map-Reduce

- Remember the first talk:
- Queries as Map- and Reduce-Jobs
Example: Map-Reduce

- Remember the first talk:
- Queries as Map- and Reduce-Jobs
- Most famous implementation:

http://hadoop.apache.org
Conclusion

- Parallel DBMS is fast, but not scalable and not fault-tolerant
Conclusion

- Parallel DBMS is fast, but not scalable and not fault-tolerant

- Map-Reduce has everything the former lacks, but is too slow
Alternative Approach

Why not join both approaches?
Our goal

A database system, called HadoopDB, which is:

• Performant
• Fault-tolerant
• Highly scalable
Our goal

A database system, called HadoopDB, which is:

- Performant
- Fault-tolerant
- Highly scalable

**Achieved via:**

- Combination of MapReduce and parallel DBMS techniques
Outline

Motivation & Goals

HadoopDB - Architecture

Hadoop Extensions

HadoopDB – Components

Benchmark
HadoopDB - Architecture

Briefly:

- Hadoop on inter-node level
- Database system on node-level
HadoopDB - Architecture

Briefly:

• Hadoob on inter-node level

• Database system on node-level

• … and several new components to achieve this
Recap: Hadoop
Now: HadoopDB
Outline

Motivation & Goals

HadoopDB - Architecture

Hadoop Extensions

HadoopDB – Components

Benchmark
Preparation

„Tune“ our hadoop elephant
Preparation

**Hadoop extensions:**

- **HIVE** for generating hadoop query plans out of SQL statements

- **HDFS** as storage unit
Hadoop Hive

- Data Warehouse System
- ETL support
- Implements a simple SQL-like language, called HiveQL
  - transforms SQL queries into Map-reduce jobs
- Supports plug-in of hand-written map-reduce operations
Hadoop Hive
Transformation Phases

1. Transform query to AST

SELECT YEAR(saleDate), SUM(revenue)
FROM sales GROUP BY YEAR(saleDate);
Hadoop Hive

Transformation Phases

2. Get data from *MetaStore*

- *Table schemes*
- *InputFormat classes*
- *Deserializer classes*
Hadoop Hive

Transformation Phases

3. Transform AST to DAG (logical query plan)
Hadoop Hive

Transformation Phases

4. Optimize plan

```
PROJECTION (sales.saleDate)
  |
  v
SELECTION(sales.saleDate=Customers.regDate)
  |
  v
  X
  |
  v
  sales
  v
Customers
```
Hadoop Hive

Transformation Phases

4. Optimize plan

Push down projection

DIAGRAM:

PROJECTION (sales.saleDate)

SELECTION (sales.saleDate = Customers.regDate)

X

sales

Customers
Hadoop Hive

Transformation Phases

4. Optimize plan

```
SELECTION(sales.saleDate=Customers.regDate)
```

```
X
```

```
PROJECTION(sales.saleDate)
```

```
sales
```

```
Customers
```

Hadoop Hive

Transformation Phases

4. Optimize plan

Replace cross product by join
Hadoop Hive
Transformation Phases

4. Optimize plan
Hadoop Hive

Transformation Phases

4. Optimize plan

Add projection

```
JOIN (sales.saleDate=Customers.regDate)
  /   \
PROJECTION (sales.saleDate) PROJECTION (Customers.regDate)
  /    \
sales       Customers
```
Hadoop Hive

Transformation Phases

5. Convert logical plan to physical plan

GroupBy(Sum(Col1), key: Col0)
  └── Projection(Year(saleDate), revenue)
      └── sales

File Sink Operator
  file: annual_sales_revenue

Select Operator
  expr: [Col[0], Col[1]]

Group By Operator
  aggr: [Sum[1]] keys: [Col[0]] mode: merge partial

Reduce Phase
  Reduce Sink Operator
    Partition Cols: Col[0]

Map Phase
  Group By Operator
    aggr: [Sum[1]] keys: [Col[0]] mode: hash

Select Operator
  expr: [Col[YEAR(saleDate)], Col[revenue]]
  [0: string, 1: double]

Table Scan Operator
  sales
Hadoop Hive

Transformation Phases

6. Save the plan graph to an XML file, start hadoop job
- Primary storage system used by hadoop systems
- Multiple replicas of data blocks
- Distributes the replicas over the cluster

→ reliable & extremely rapid computations
Outline

Motivation & Goals

HadoopDB - Architecture

Hadoop Extensions

HadoopDB – Components

Benchmark
HadoopDB - Components
HadoopDB - Components

**Database connector**

- Implements hadoop inputFormat class
- Connection between hadoop and databases on nodes
- until now: implemented for mysql and postgresql
HadoopDB - Components

Catalog

- Contains metadata over the DB
- Needed for query planning
HadoopDB - Components

Data Loader

- Loads Data from a file into the database:
  1. Global repartitioning regarding node size using a key
  2. Again partitioning the node parts into smaller chunks
  3. Loading the chunks via bulkloading
HadoopDB - Components

**SMS Planner**

- SQL to MapReduce to SQL Planner
- Extension of HIVE
SMS-Planner - changes

Hive Phases-Recap

1) Transform query to AST
2) Get data from Metastore
3) Transform AST to DAG
4) Optimize plan
5) Convert logical to physical plan
6) Save & execute plan
SMS-Planner - changes

**Hive Phases-Recap**

1) Transform query to AST
2) Get data from Metastore
3) Transform AST to DAG
4) Optimize plan
5) Convert logical to physical plan
6) Save & execute plan

Add data from HadoopDB Catalog
SMS-Planner - changes

Hive Phases-Recap

1) Transform query to AST
2) Get data from Metastore
3) Transform AST to DAG
4) Optimize plan
5) Convert logical to physical plan
6) Save & execute plan
Plan transformation

- Retrieve accessed data fields and partitioning keys
- Plans that can be executed locally on the node are converted back to SQL and pushed into the database system
SMS-Planner - Example

SELECT YEAR(saleDate), SUM(revenue) FROM sales GROUP BY YEAR(saleDate);
SELECT YEAR(saleDate), SUM(revenue) FROM sales GROUP BY YEAR(saleDate);
**SMS-Planner - Example**

```
SELECT YEAR(saleDate), SUM(revenue)
FROM sales GROUP BY YEAR(saleDate);
```
```sql
SELECT YEAR(saleDate), SUM(revenue)
FROM sales GROUP BY YEAR(saleDate);
```
HadoopDB - Architecture
Outline

Motivation & Goals
HadoopDB - Architecture
Hadoop Extensions
HadoopDB – Components
Benchmark
The final question

Did we reach our goal?
Benchmark

The Opponents

1. vertica:
   - Column-store parallel DB
   - Cloud computing
   - Operates on compressed data
Benchmark

The Opponents

2. dbms-X:
   - Row-store parallel DB
   - **NO** Cloud computing
   - Also operates on compressed data
Benchmark

The Opponents

3. hadoop:
   - The well-known map-reduce implementation
   - One time with HIVE generated plans
   - The other time with hand-written map-reduce operations
Benchmark

Tests-Tasks

1. **Load**: load a greater amount of data
2. **Grep**: find a string pattern, containing of 3 characters
3. **Selection**: execute a simple SQL selection query
4. **Aggregation**: execute an aggregation query
Benchmark

Tests-Tasks ctd.

5. **Join**: execute a Join query

6. **UDF aggregation**: apply a user defined function (UDF) and aggregate afterwards

7. **fault-tolerance**: simulate failure of a single node

8. **heterogenity**: slow down a single node
Benchmark

Load
Benchmark

Grep

[Diagram showing benchmark results for different systems and node counts]
Benchmark

Selection

![Chart showing benchmark results for selection queries with different number of nodes and systems. The chart includes bars for Vertica, DB-X, HadoopDB, HadoopDB Chunks, and Hadoop, with performance metrics measured in seconds. The x-axis represents the number of nodes (10, 50, 100), and the y-axis represents time in seconds.]
Benchmark

Aggregation
Benchmark

Join

![Graph showing benchmark results for Join operation with different numbers of nodes and database systems. The graph compares Vertica, DB-X, HadoopDB, and Hadoop. The results indicate varying performance times across different systems and node counts.](image)
Benchmark

UDF Aggregation
Benchmark

Fault Tolerance & Heterogeneity

% slowdown = \( \frac{(n - f)}{n} \times 100 \) or \( \frac{(n - s)}{n} \times 100 \)

- n: normal execution time
- f: execution time with single node failure
- s: execution time with a single slow node
Conclusion

- HadoopDB is much slower than a commercial DBMS
- Though it can cope better with heterogenous networks and node failures, the total processing time remains a lot larger
Conclusion

**But**

- it is lot faster than the standard Hadoop
- uses a quite slow database implementation (postgresql)
- Just a „student“ project, a proof-of-concept

→ *will become a lot faster when development makes process*
THANKS FOR LISTENING.
Questions ?