A Comparison of Approaches to Large-Scale Data Analysis

Andrew Pavol, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, Michael Stonebraker
Published in SIGMOD ‘09

Presented by Mingxiao (Michelle) Song, Huaijin (Tony) Tu, Anh Nguyen, Natasha Mohanty

Motivation and Overview

Rise of Clustering Computing

Large number of low-end servers instead of deploying a smaller set of high-end servers.

Clustering Programming Tools

E.g. MapReduce, simple tool to express tasks for sophisticated distributed system.

Large Scale Data Analysis

Dividing any data set to be utilized into partitions, which are allocated to different nodes to facilitate parallel processing.

Q: How to compare the MapReduce approach with parallel SQL database management systems?

A: Evaluate **MapReduce** and **parallel SQL DBMS** in terms of *performance* and *development complexity.*

- Define benchmark with parallel processing tasks
- Summarize trade-offs that future systems should take from both kinds of architectures
  - schema format
  - indexing & compression optimization
  - How data is distributed
  - Query execution strategies

MapReduce

Map

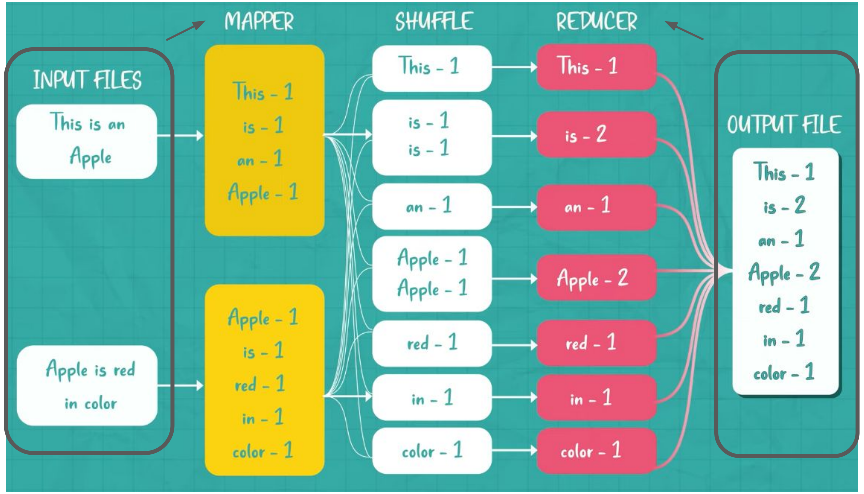
- Reads set of records from input file
- Conducts **filtering/transformation**
- Outputs set of intermediate records in the form of **key-value** pairs

Split

- **Partitions** key-value pairs into R disjoint buckets by applying hash function on keys
- **Writes** map bucket to the processing node's local disk

Reduce

- Executes R Reduce instances
- Related key-value pairs transferred over network from the Map node's local disk
- **Processes/combines** records assigned to it
- Write records to an output file



Simplicity

MR scheduler decide:

- how many Map/Reduce instances to run;
- how to allocate them to available nodes

1. Map Reduce explained with example: https://www.youtube.com/watch?v=cHGaQz0E7AU&ab_channel=ByteMonk

Parallel DBMS

Database systems capable of running on clusters of shared nothing nodes. (since 1980s)
Example task: *filter* records in table T1 based on a predicate, along with a *join* to a second table T2 with an *aggregate* computed on the result of the join.

01

Most tables are partitioned over the nodes in a cluster.

02

The system uses an optimizer that translates SQL commands into a query plan whose execution is divided amongst multiple nodes.

Perform filter sub-query at nodes that already stored T1 (similar to Map function)

Join algorithms based on the size of data table T2

T2 is small

- Replicate T2 on all nodes
- Execute join in parallel at all nodes
- Compute aggregate at each node
- Final roll up

T2 is large

- Distribute T2 content across multiple nodes (different sets of attributes)
- Hash both T2 and the filtered T1 on the join attribute (similar to Split)
- Compute aggregate at each node
- Final roll up

Architectural Elements

- Schema Support
- Indexing
- Programming Model
- Data distribution
- Execution strategy
- Flexibility
- Fault tolerance

Schema Support

MapReduce (MR)	Parallel DBMS
Does not require data to adhere to a predefined schema.	Requires data to fit into the relational model of rows and columns.
Programmers have the freedom to structure data in any format.	Enforces data integrity and schema constraints automatically.
Custom parsers often needed for complex data structures.	Simplifies data sharing and reuse across applications.

Indexing

MapReduce (MR)	Parallel DBMS
Does not provide built-in indexes.	Utilizes indexing (e.g., hash or B-tree indexes) to accelerate data access.
Programmers must implement any required indexing within their applications.	Supports multiple indexes per table. Query optimizer decides the best index to use for each query.

Programming Model

MapReduce (MR)	Parallel DBMS
Low-level, procedural programming model.	High-level, declarative SQL programming model.
Consists of Map and Reduce functions for processing key/value data pairs.	Abstracts away the underlying data storage and retrieval mechanisms.
Requires explicit algorithms for data manipulation. *Pig, Hive improve on this drawback.	Easier for users familiar with SQL; less flexibility for procedural programming.

Data Distribution

MapReduce (MR)	Parallel DBMS
“Share nothing” architecture	“Share nothing” architecture
Data is manually distributed across nodes.	Automatically manages data distribution.
Requires programmers’ effort to manage data distribution and processing.	Uses a parallel query optimizer to manage data distribution and query execution.
Data shipped from Mapper to Reducer.	Program “finds” data by optimizer, pushing down when possible

```
CREATE VIEW Keywords AS
SELECT siteid, docid, word, COUNT(*) AS wordcount
  FROM Documents
 GROUP BY siteid, docid, word;
SELECT DISTINCT siteid
  FROM Keywords
 WHERE (word = 'IBM' OR word = 'Google') AND wordcount > 5;
```

Execution Strategy

MapReduce (MR)	Parallel DBMS
Pull-based model for data transfer between Map and Reduce phases.	Push-based model, avoiding materialization of intermediate results.
Data transfer between Map and Reduce phases can be a bottleneck due to disk I/O and network congestion.	Optimizes execution plans globally, minimizing data transfer.

Flexibility

MapReduce (MR)	Parallel DBMS
Highly flexible in processing unstructured data.	Primarily designed for structured data processing.
Supports a wide range of custom data processing tasks beyond traditional database queries.	Less flexible in handling unstructured data compared to MR.

Fault tolerance

MapReduce (MR)	Parallel DBMS
Highly fault-tolerant with automatic task retries and data replication.	Provides fault tolerance mainly through data replication.
Designed to handle failures gracefully during query execution.	May require complete query restarts in the event of node failures (costly when query takes long to run).

Configuring Tested Systems

1 **Hadoop**
Framework for distributed storage & datasets

2 **DBMS-X**
Parallel-SQL DBMS storing row-oriented data

3 **Vertica**
Analytics parallel DBMS for large warehouses

The Original MR Task

The **“Grep Task”** represents a multifaceted task for storing large subsets of data programs where it scans through *100-byte* records looking for a *three-character* pattern. To measure scaling performance, the task is executed on *two* unique datasets

- Creation statement where input data is stored as text files

```
CREATE TABLE Data (
   key VARCHAR(10) PRIMARY KEY,
   field VARCHAR(90) );
```

Subsets of “Grep Task”

01

Data Loading focuses on the time taken for to load test data

02

Task Execution measures how well each system scales as number of available nodes increases

Data Loading - System Method

DBMS-X and Vertica (DBMS's)

- Loading process involves two phases: executing the *load-sql* command in parallel and delimiting the data by a special character
- The system must redistribute the tuples to other clusters because the data generator generates random keys
- After loading the data, the administrative command is executed to reorganize the data

Hadoop (MapReduce)

- Two ways to load: either through the command line utility to upload files or custom data loader program to write data to internal API
- Files on each node are loaded in parallel as plain-text
- Allows for MR programs to access data using Hadoop TextInputFormat data format

Data Loading - Performance Comparison

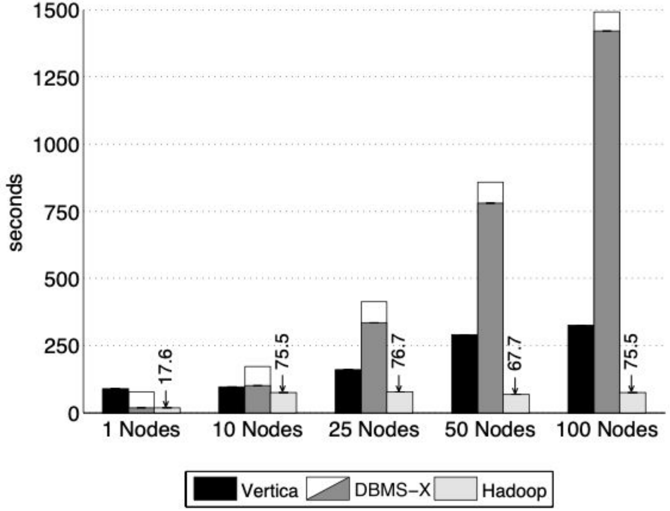


Figure 1: Load Times – Grep Task Data Set (535MB/node)

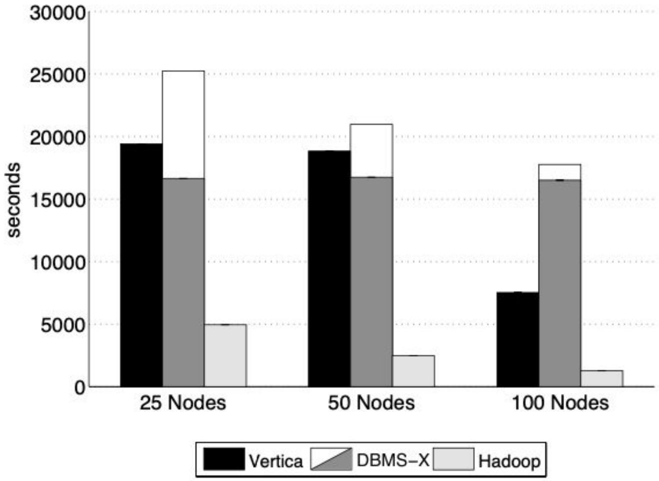


Figure 2: Load Times – Grep Task Data Set (1TB/cluster)

Task Execution - SQL Statement

The MR Program contains a singular Map function splitting a record into its *key/value* pairs. By performing a sub-string match for a search value. If a search pattern is found, then the Map function outputs an input *key/value* pair to HDFS

- The following **SQl Statement** focuses on pattern search for a particular field

```
SELECT * FROM Data WHERE field LIKE '%XYZ%';
```

Task Execution - Performance Comparison

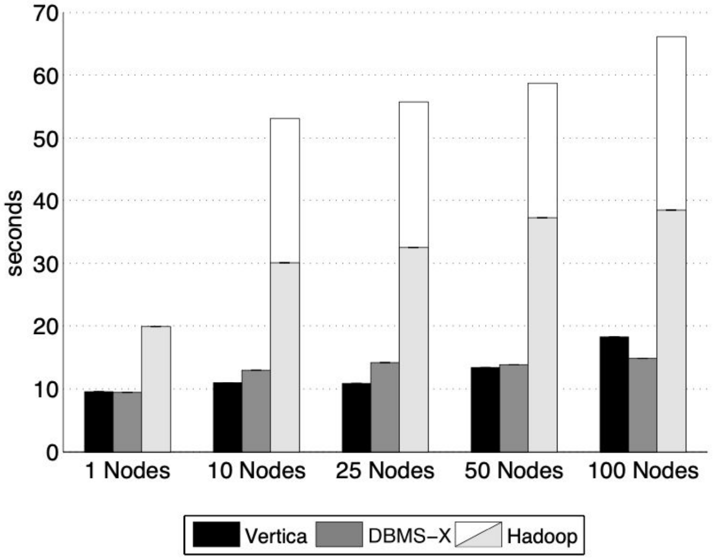


Figure 4: Grep Task Results – 535MB/node Data Set

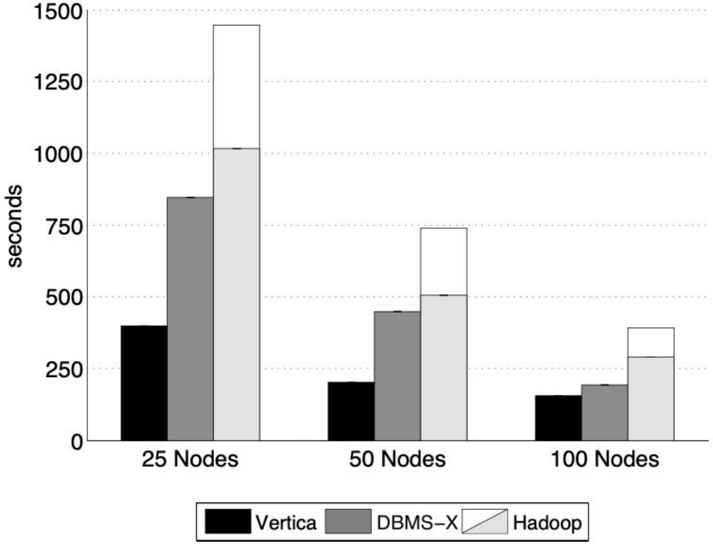
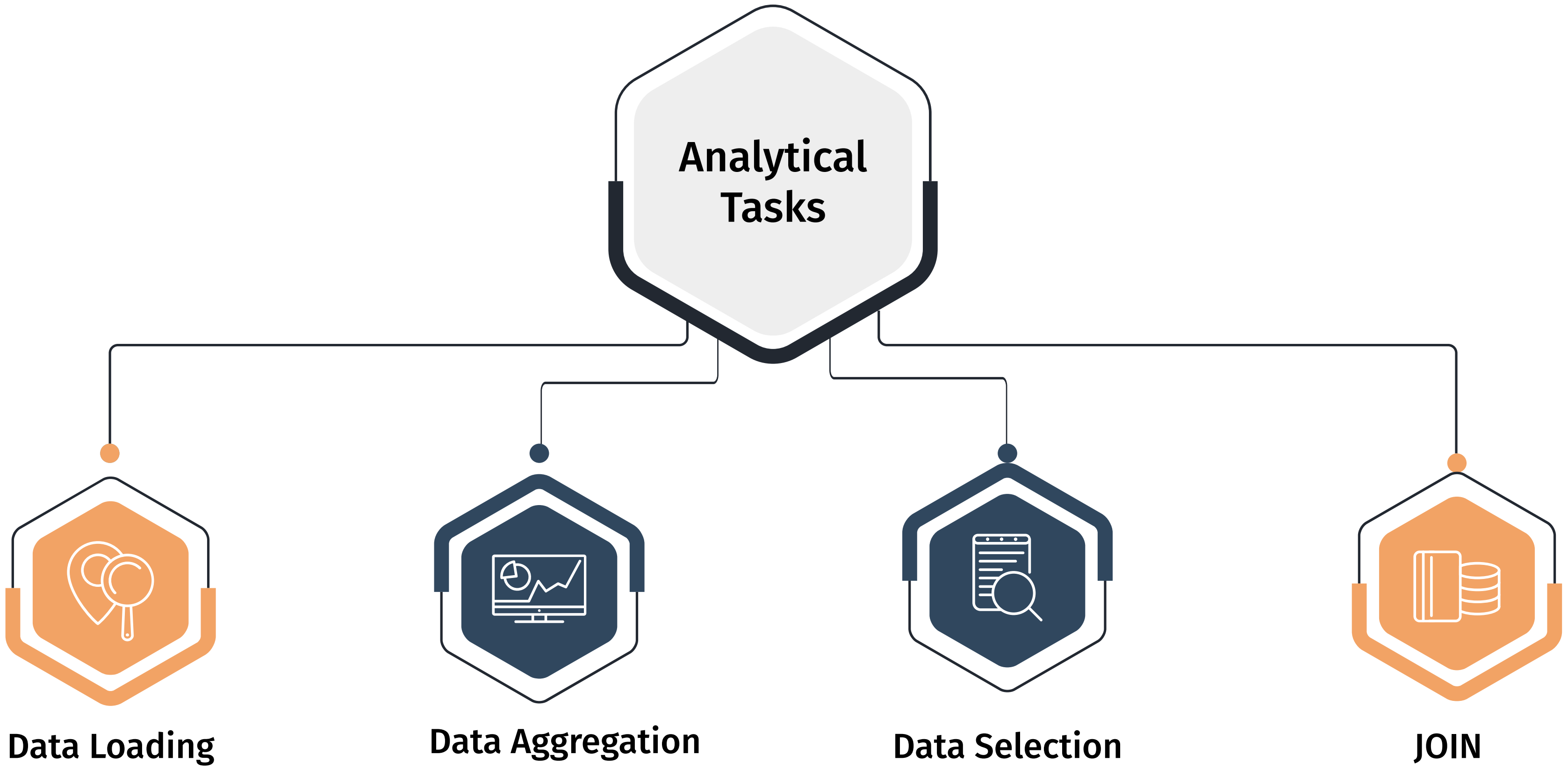


Figure 5: Grep Task Results – 1TB/cluster Data Set


Analytical Tasks
Data Loading
Data Aggregation
Data Selection
JOIN

Data Loading - UserVists and Ranking

DBMS-X and Vertica (DBMS's)

- Use UDF that processes the documents on each node at runtime and loads the data into a temporary table
- Significantly faster with the ability to directly load structured data

Hadoop (MapReduce)

- Loads the Documents files into its internal storage system
- Requires custom data loaders to modify the datasets for MR compatibility
- This process is manual

Data Selection - DBMS

DBMS-X and Vertica (DBMS's)

- Execute a simple SQL query

```
SELECT pageURL, pageRank
  FROM Rankings WHERE pageRank > X;
```

Data Selection - MR

Map Filters records based on a threshold value (e.g., pageRank > x)

Reduce No reduce function is needed in this case

Data Selection - Performance Breakdown

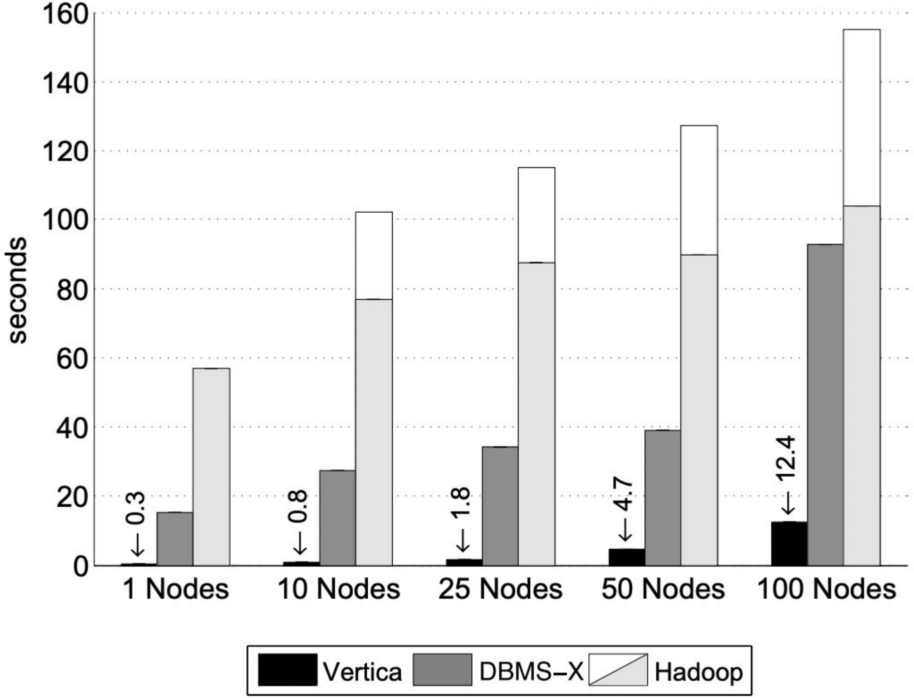


Figure 6: Selection Task Results

Data Aggregation - DBMS

DBMS-X and Vertica (DBMS's)

- Execute a simple SQL query to group and perform summation

```
SELECT sourceIP, SUM(adRevenue)
  FROM UserVisits GROUP BY sourceIP;
```

Data Aggregation - MapReduce

Map

Outputs the sourceIP field and the adRevenue field as a new key/value pair

Reduce

Adds together all of the adRevenue values for each sourceIP and then outputs the sourceIP and its revenue total

Data Aggregation - Performance Comparison

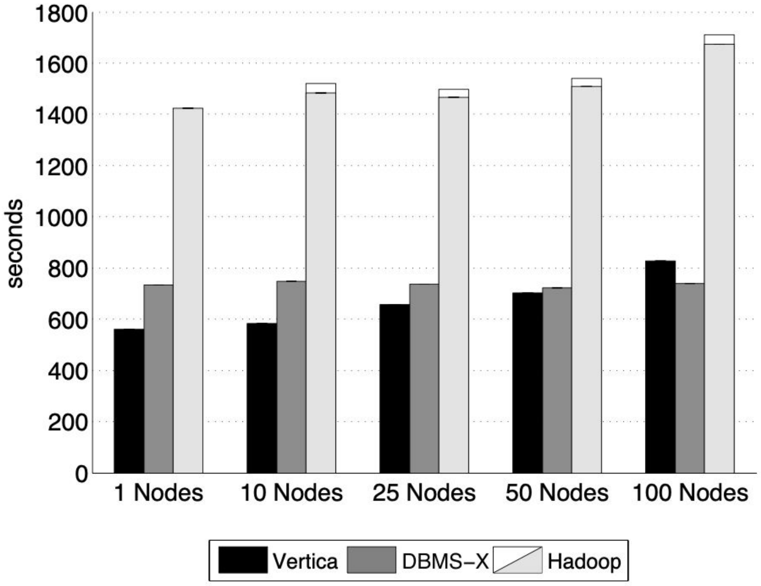


Figure 7: Aggregation Task Results (2.5 million Groups)

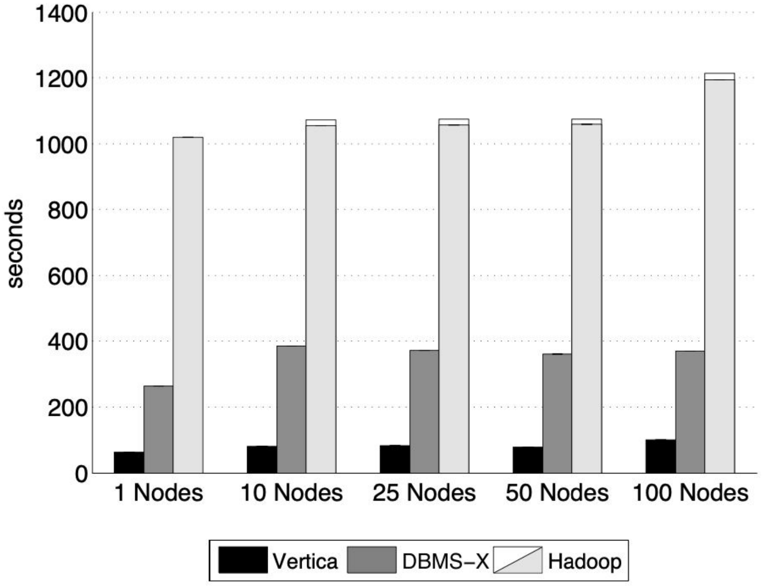


Figure 8: Aggregation Task Results (2,000 Groups)

Join Task - DBMS

DBMS-X and Vertica (DBMS's)

- SQL queries:

```
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 Date('2000-01-15')
                        AND Date('2000-01-22')
 GROUP BY UV.sourceIP;

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

Join Task - MapReduce

1

Filtering Records

2

Perform Actual Join

3

Aggregating results

Join Task - Performance Comparison

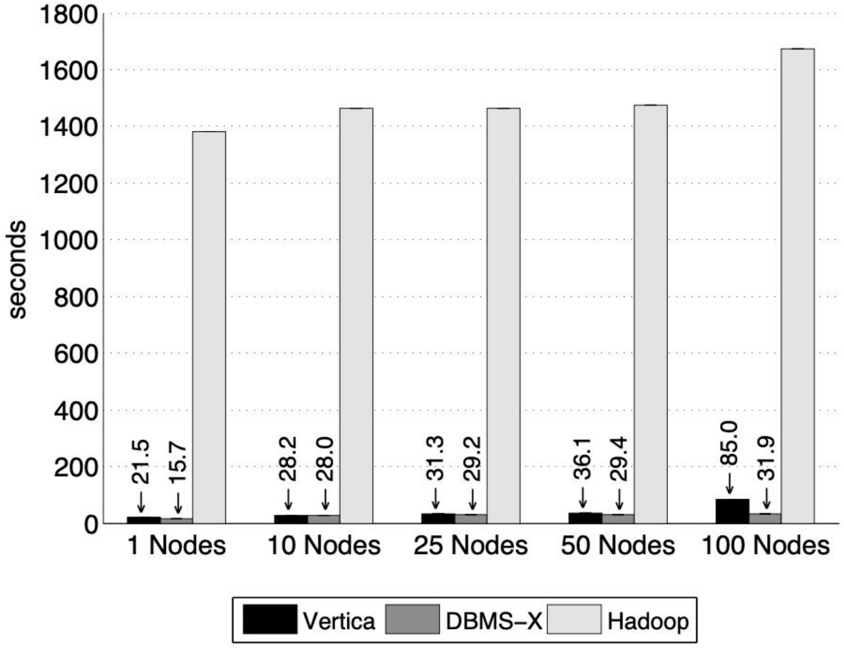


Figure 9: Join Task Results

Conclusion

- Parallel database systems demonstrated significant performance advantage compared to Hadoop
  - DBMS-X was 3.2x faster than MR & Vertical was 2.3x than DBMS-X
  - Performance advantage due to numerous current technologies
- Future areas to focus on:
  - Evaluating the performance penalty in parallel computing scope
  - Understanding the schema implementation of MR

Study Question

Question 1: Considering the performance advantages of parallel DBMS over MapReduce for the tasks in the benchmark, what are the potential implications for the future development of data processing frameworks, and how might MapReduce adapt to remain competitive?

Question 2: Given the ease of use and setup of MapReduce compared to parallel databases, as noted in the paper, what strategies could parallel databases employ to improve their usability and reduce setup complexity, and how might this impact their adoption in various industries?