Comparison of Parallel DB and MapReduce

MapReduce: A Flexible Data Processing Tool

Grey Beards:“ MapReduce is a major step backwards”
Young Turks:” No, it’s because you have so many misconceptions about MapReduce.”

Original Slides Author: John Kubiatowicz
Modified by: Jingxuan Huang (Carol)
Presentor: Jingxuan Huang (Carol)
Discussion Leader: Ehsan Soltan Aghai (Ehsan)

MapReduce vs Parallel DB: are they comparable?

“Though it may seem that MR and parallel databases **target different audiences**, it is in fact **possible to write almost any parallel processing task** as either a set of database queries or a set of MapReduce jobs”

Similarity

1. “shared nothing “ architecture
2. achieve parallelism by dividing any data set to be utilized into partitions

Difference

	Parallel DB	MapReduce
Schema Support	Yes	No
Built-in Index	Yes	No
Programming Model	Declarative (SQL)	Procedural (C/ C++/ Java)
Flexibility	Not as high	High
Execution Strategy	Push	Pull
Fault Tolerance	Not as good	Good

Difference

	Parallel DB	MapReduce
Configuration	Complex; one-shot	Easy; for each task
Start-up	Warm	“Cold start”
Compression	Save time and space	Not improve performance
Loading	Slow, many pre-processing	Easy and fast

Schema Support

MapReduce

- No schema required
- Flexible, no need to predefine schema
- Bad if data are **shared** by multiple applications. Must address data syntax, consistency, etc.
- Cannot ensure **integrity constraints** (e.g., employee salaries must be non negative); vulnerable to bad data

Parallel DBMS

- Relational schema required
- Good if data are shared by multiple applications

Programming Model & Flexibility

MapReduce

- 2 functions: Map and Reduce
- little data independence: presenting algorithms for data access

"We argue that MR programming is somewhat analogous to Codasyl programming...was criticized for being "the assembly language of DBMS access""

- better generality

Parallel DBMS

- declarative language like SQL
- insufficient expressive prowess
- SQL can be hard to use for people brought up programming in procedural languages

Indexing

MapReduce

- No built-in indexes
- Programmers can implement their own index support in Map/ Reduce code (not easy)
- But hard to share the customized indexes in multiple applications

Parallel DBMS

- All modern DBMSs use Hash/b-tree indexes to accelerate access to data

MapReduce’s Defence

- An index can be added to each database, which can be used as an input to MapReduce.
- When MR reads from Bigtable, can read only a sub-range or selected columns (to avoid full scan)

Data Distribution

MapReduce

- need to manually compute statistics before utilizing them

Parallel DBMS

- Leverage the knowledge of data distribution to schedule and minimize the amount data transmitted over the network
- Automatic query optimization

Execution Strategy & Fault Tolerance

MapReduce

- **Pull**: seek data for computation
- Intermediate results are saved to local files
- When multiple Reducers are reading local files from Map workers, there could be large numbers of disk seeks, leading to poor performance.
- If a node fails, restart the task on an alternative node (without aborting the whole computation)

Parallel DBMS

- **Push**: send computation to data
- Avoid Intermediate results, push across network
- If a single node fails, must re-run the entire query

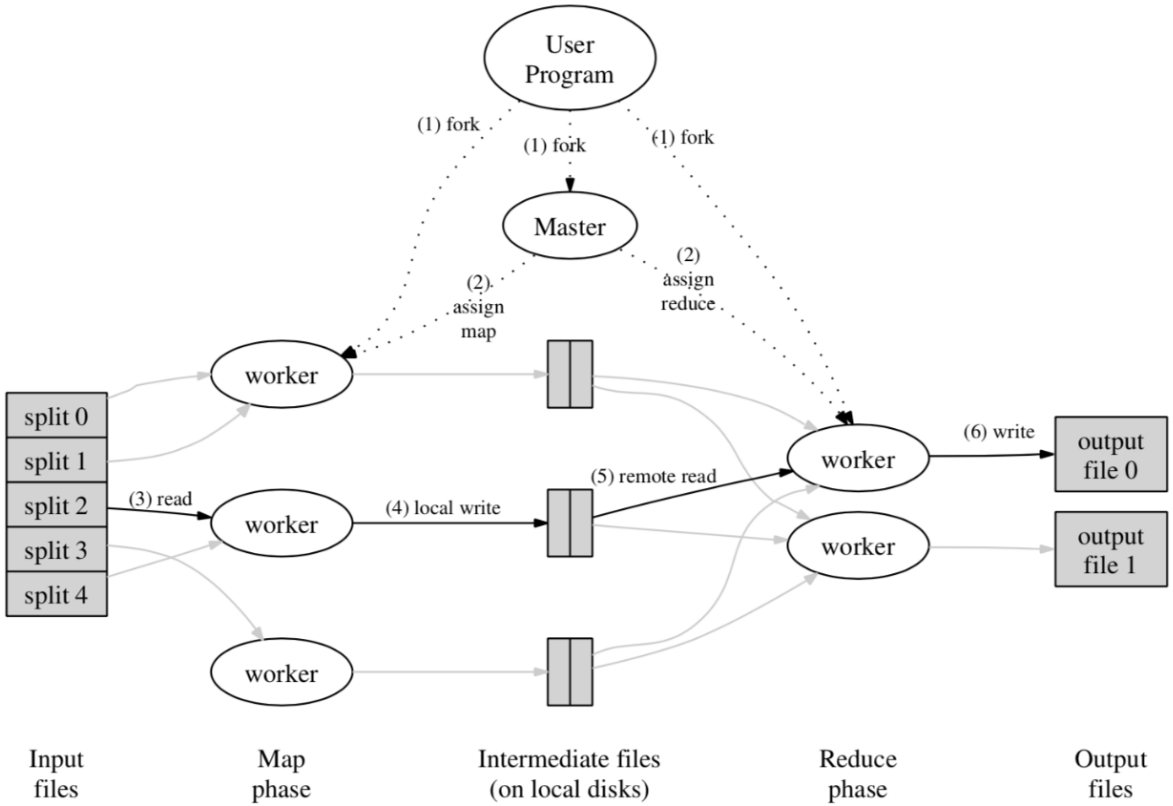

Figure 1: Execution overview

MapReduce's Defence

- They chosed Pull model due to the **fault-tolerance properties** required by Google's developers
- Fault-tolerance being more important in the future

Discussion Question

Rank the following features in large-scale data analysis from the most important one to the least:

- Schema support
- Indexing
- Programming model
- Data distribution
- Execution strategy
- Flexibility
- Fault tolerance

Discussion Question

	G1	G2	G3	G4
Schema support	6	7	6	2
Indexing	1	2	4	7
Programming model	5	3	5	6
Data distribution	2	6	2	4
Execution strategy	4	4	3	3
Flexibility	7	1	7	5
Fault tolerance	3	5	1	1

Performance Benchmarks

Benchmark Environment: 100-node cluster (controversial)

Tested Systems:

- MapReduce framework: Hadoop
- Parallel DB: DBMS-X (an unidentified commercial database system), Vertica

Data Loading

Hadoop: load to HDFS as plain text (*in parallel*)

DBMS-X: two phases

- read from the local file system (*sequentially*)
- reorganize data on each node (e.g., compress data, build index) (*in parallel*)

Vertica: load data in parallel and automatically sorted and compressed

Data Loading

Data Inputs: (2 Data sets)

1. Scaleup: Fix the size of data per node (535MB/node), add nodes and data
2. Speedup: Fix the total data size (1TB), add nodes

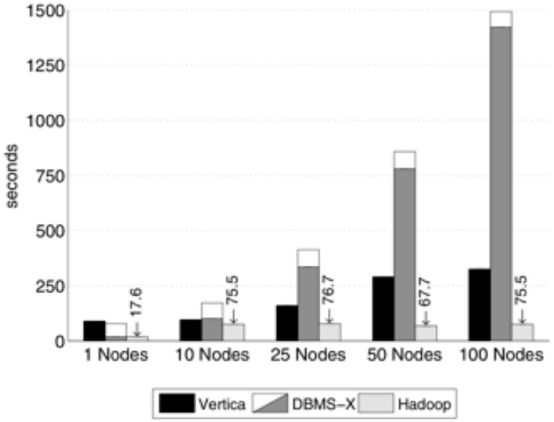

Figure 1: Load Times – Grep Task Data Set (535MB/node)

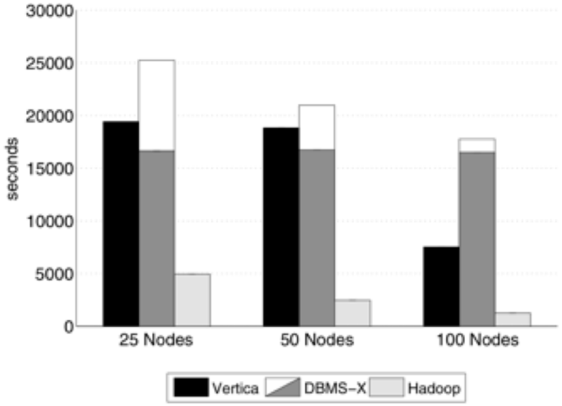

Figure 2: Load Times – Grep Task Data Set (1TB/cluster)

Performance Benchmarks

Tasks:

- Original MR task (Grep: *globally search a regular expression and print*)
- Analytical Tasks (related to HTML document processing)
  - ❖ Selection
  - ❖ Aggregation
  - ❖ Join
  - ❖ User-defined-function (UDF) aggregation

For each task, Hadoop needs to do an additional Reduce job to combine the output into a single file (which is argued unnecessary in the 2nd paper)

Grep Task Execution Performance

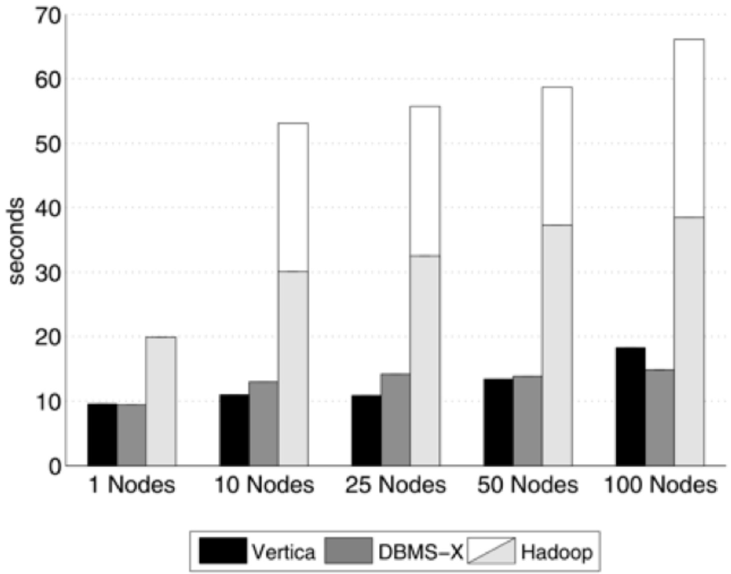

Figure 4: Grep Task Results – 535MB/node Data Set

(Fix the size of data per node)

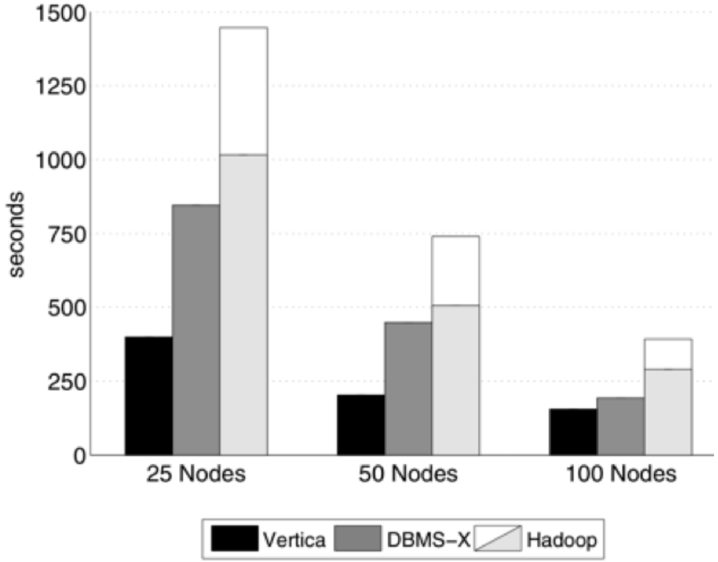

Figure 5: Grep Task Results – 1TB/cluster Data Set

(Fix the total data size)

MapReduce’s Defence

- High start-up overhead is due to the immature implementation, not fundamental differences in programming models.
    - Google has started optimizing performance

Select Task Performances

- Find the pageURLs in the rankings table (1GB/node) with a pageRank > threshold

SQL:

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

MR:

single Map, no Reduce

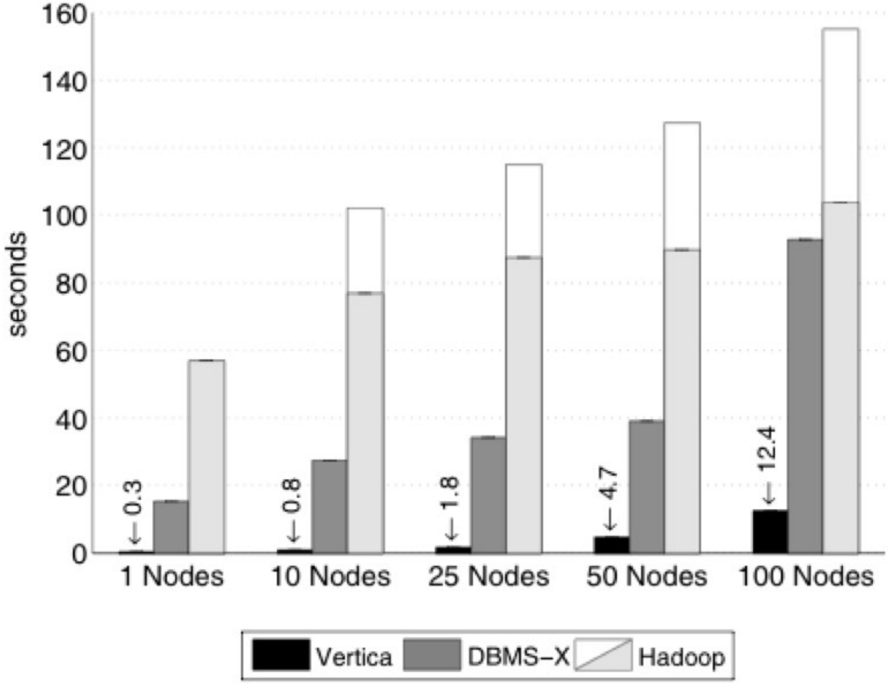

Figure 6: Selection Task Results

MapReduce's Defence

- To avoid a full scan, the input of MapReduce can be a database with an index that provides efficient filtering or an indexed file structure.
    - *(still rely on db to solve its own issue)*

Aggregation Performances

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

2 versions, to test the effect of #groups on query performance

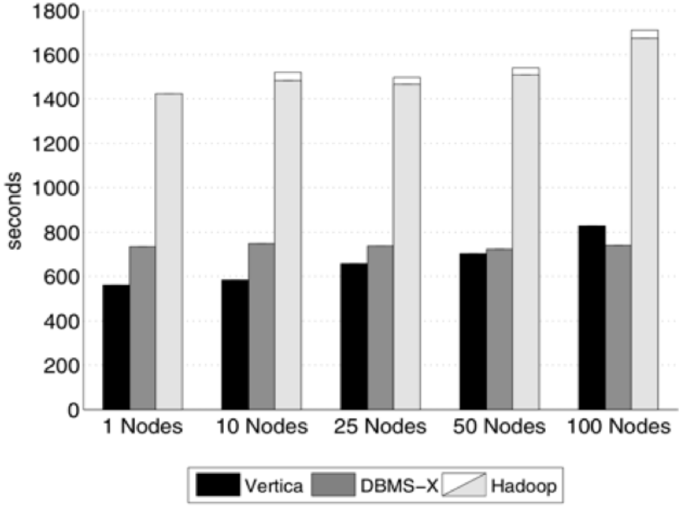

Figure 7: Aggregation Task Results (2.5 million Groups)

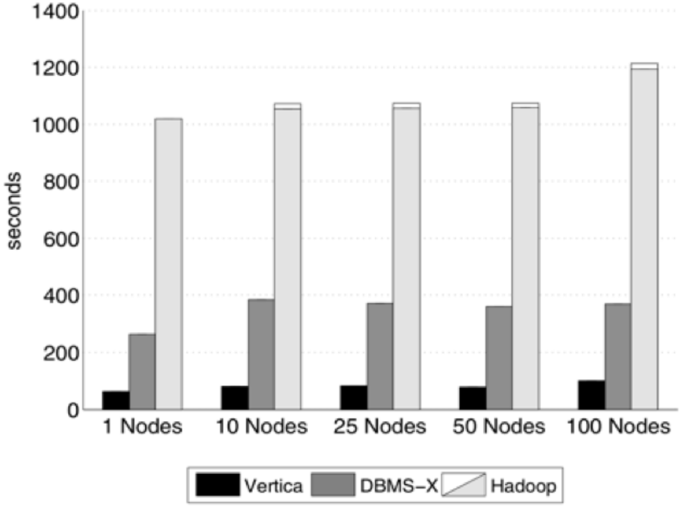

Figure 8: Aggregation Task Results (2,000 Groups)

Join Performances

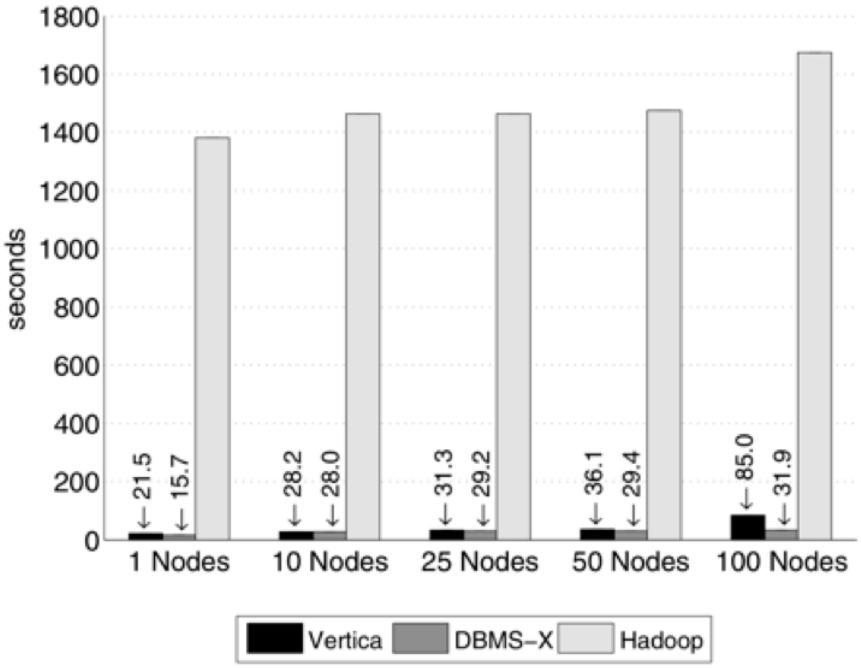

Figure 9: Join Task Results

UDF Aggregation Performances

Count the number of inlinks for each document (~PageRank calculations)

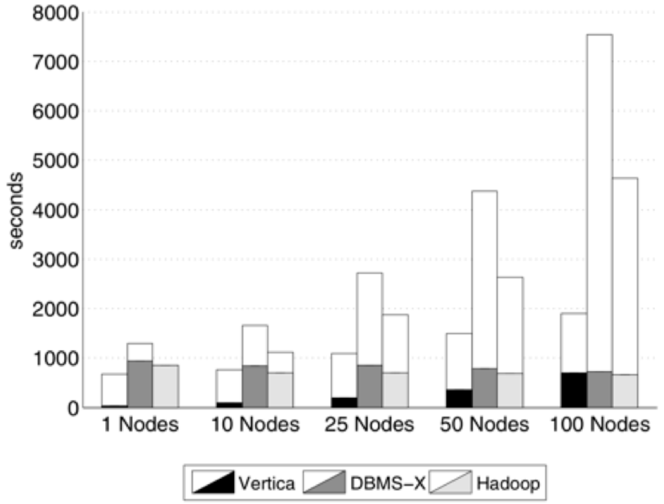

Figure 10: UDF Aggregation Task Results

MapReduce's Defence: Why Hadoop performs so bad in comparison paper?

- It used textual format as input, whereas at Google they use **Protocol Buffer format** to read and write data. It will dramatically improve performance (e.g., for parsing input, 20 nanoseconds per record as compared to the 1,731 nanoseconds)
- Reading unnecessary data (select, aggregation, join)
- No need for merging results
- Tons of loading time wasted in parallel DBMSs

MapReduce's Defence: Why is MapReduce better?

- Heterogeneous system: a mix of storage systems
  - MR provides a simple model for analyzing data in heterogenous systems.
- Easy and fast loading: Especially because "Data sets are often generated, processed once or twice, and then **<u>discarded</u>**"
  - *"it is possible to run 50 or more separate MapReduce analyses before it is possible to load the data into a database and complete a single analysis"*
- Supports complex functions (compared to the awkward UDF)

Discussion Question

MapReduce misconceptions:

- ❖ Why are there many "incorrect understandings" on MapReduce?
  - ➢ MapReduce cannot use indices and implies a full scan of all input data.
  - ➢ MapReduce input and outputs are always simple files in a file system.
  - ➢ MapReduce requires the use of inefficient textual data formats.
- ❖ It is obvious that the comparison paper authors have internal biases toward MapReduce. If you are a critic of a method, how can you prove your point while maintaining a neutral stance? (Jeffrey)
- ❖ Since industry is not very transparent about their work and research, there will always be miscommunication between academia and industry. What can people do to alleviate such miscommunication? (Jianhao)

History Repeats Itself: Sensible and NonsenSQL Aspects of the NoSQL Hoopla

-- "Human's demand of query type is changing in web 2.0."

-- "Not everything needs to be done differently just because it is supposedly a very different world now!"

--- C.Mohan *(who proposed ARIES)*

Slides Made By: Yisheng Zhu (Ethan)
Modified by: Jingxuan Huang (Carol)
Presentor: Jingxuan Huang (Carol)
Discussion Leader: Ehsan Soltan Aghai (Ehsan)

Why RDBMSs are inadequate nowaday?

In certain types of applications, typically **Web 2.0** ones, for which RDBMSs were found to be inadequate:

- ❖ Data is less structured and the structure changes a lot.
- ❖ To Become a master of RDBMS, you need learn SQL
- ❖ Response times are critical
- ❖ Lower consistency requirements
- ❖ Types of query has changed: simple data accesses but large volumes of data
- ❖ Graceful ways of handling failures of individual nodes
- ❖ Commodity servers

Observed Problems of NoSQL

- The importance of thinking about locking, storage management and recovery concurrently, instead of adding these functionality later which would be very hard (lessons from ARIES)
- Goodness of standards are forgotten in the context of NoSQL systems.
- Forgot the benefits of high level languages and data independence.
- Indexing should not be lost.
- Data model of NoSQL is not necessarily simpler. Varying data models can be a nightmare for data migration.
- Not supporting ACID transaction functionality is oversimplification.

Discussion Question

- This paper compared similarity and new requirements of NoSQL over RDBMS in indexing, data models, document stores and transactions. Think of other features that NoSQL might distinct from RDBMS.
- In what realistic cases might the limitations in NoSQL in not applying transactions (and its ACID functionality) come back to hurt a company? (Michael)