Comparison of Parallel DB and MapReduce

MapReduce: A Flexible Data Processing Tool

Grey Beards: “MapReduce is a major step backwards”[1]
Young Turks: “No, it’s because you have so many misconceptions about MapReduce.”[2]

Original Slides Authors: John Kubiatowicz
Modified by: Jingxuan Huang (Carol), Juntong
Presenter: Juntong
Discussion Lead: Soo Yee

1. Blogpost: https://homes.cs.washington.edu/~billhowe/mapreduce_a_major_step_backwards.html
2. MapReduce: A flexible Data Processing Tool

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”

Similarities

1. Both run on “shared nothing” architecture
2. Both achieve parallelism by partitioning the datasets

Differences

	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

Differences

	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

Discussion Question (Groups of 3)

	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

	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

What are the use cases these systems are best suited for? Why?

Schema Support

MapReduce

- No schema required
- Flexible, no need to predefine schema
- If data are **shared** by multiple programs, developers must agree on a schema
- Cannot ensure **integrity constraints** (e.g., salaries must be non-negative); it's the programmers' job to adhere to them.

Flexible when there's no sharing

Parallel DBMS

- Schema required
- Schema is separate from the application
- Constraints enforced automatically

Good when sharing is needed

Schema Support

MapReduce

- No schema required
- Flexible, no need to predefine schema
- If data are **shared** by multiple programs, developers must agree on a schema
- Cannot ensure **integrity constraints** (e.g., salaries must be non-negative); it's the programmers' job to adhere to them.

Flexible when there's no sharing

Parallel DBMS

- Schema required
- Schema is separate from the application
- Constraints enforced automatically

Good when sharing is needed

MapReduce's Defence:

- MapReduce can read and write data with schema defined with Google's Protocol Buffer.
- Example:

```
message Rankings {
  required string pageurl = 1;
  required int32 pagerank = 2;
  required int32 avgduration = 3;
}
```

Indexing

MapReduce

- No built-in indexes, leading to a full scan for all input
- Building custom indexes is hard, as the framework's data fetching mechanisms must be changed
- Hard to share the customized indexes between multiple programmers

Parallel DBMS

- All modern DBMSs have hash/B-tree indexes to accelerate access to data

Indexing

MapReduce

- No built-in indexes, leading to a full scan for all input
- Building custom indexes is hard, as the framework's data loading mechanisms must be changed
- Hard to share the customized indexes between multiple programmers

Parallel DBMS

- All modern DBMSs have hash/B-tree indexes to accelerate access to data

MapReduce's Defence:

- A full scan of all input data is not always needed; MapReduce can use the index provided by the input interface.
- For example, the input data may be:
  - Files with natural indexes (e.g., timestamps in log file names)
  - A database with an index (e.g., BigTable)

Programming Model

MapReduce

- **Procedural:** Map and Reduce
- Analogous to **Codasyl** – “the assembly language of DBMS access”

Parallel DBMS

- **Declarative (e.g., SQL)**

This comparison resembles the debate between **Codasyl** and **Relational**

- “Programs in high-level languages, such as SQL, are easier to write, easier to modify, and easier for a new person to understand.”

Data Distribution

MapReduce

- A naïve implementation of query could lead to unnecessary data transfer between nodes

Parallel DBMS

- Query optimizers can automatically rewrite queries to reduce network I/O

Execution Strategy

MapReduce

- **Pull**: Reducers pull data for computation from Mappers
- Materializes intermediate results on disks
- When multiple Reducers are reading files from the same Mapper, there could be large numbers of disk seeks

.

Parallel DBMS

- **Push**: send computation to data
- Don’t store intermediate results on disks

Execution Strategy

MapReduce

- **Pull**: Reducers pull data for computation from Mappers
- Materializes intermediate results on disks
- When multiple Reducers are reading files from the same Mapper, there could be large numbers of disk seeks

.

Parallel DBMS

- **Push**: send computation to data
- Don't store intermediate results on disks

MapReduce's Defence:

- The Pull model was due to fault-tolerance properties required by Google's developers.
  - Fault-tolerance would be more important in the future due to growing dataset sizes.
- Implementation tricks can mitigate these costs

Flexibility

SQL does not facilitate the desired generality that MR provides, but…

- Major DBMSs now provide support for user-defined functions, stored procedures, and user-defined aggregates in SQL, increasing their flexibility

MapReduce's Defence:

- In many cases, the function is too complicated to be expressed easily in a SQL query.
- UDF support was still either buggy (in DBMS-X) or missing (in Vertica)

Fault Tolerance

MapReduce

- If a node fails, just restart the task on an alternative node (without aborting the whole computation)

Parallel DBMS

- If a single node fails, must re-run the entire query

Discussion Question (Groups of 4)

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

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

Performance Benchmarks

Benchmark Environment: 100-node cluster

- “Since few data sets in the world even approach a petabyte in size, it is not at all clear how many MR users really need 1,000 nodes.”

Performance Benchmarks

Benchmark Environment: 100-node cluster

- “Since few data sets in the world even approach a petabyte in size, it is not at all clear how many MR users really need 1,000 nodes.”

MapReduce’s Defence:

- “There are already more than a dozen distinct data sets at Google more than 1PB in size and dozens more hundreds of TBs in size that are processed daily using MapReduce. “

Performance Benchmarks

Benchmark Environment: 100-node cluster

- “Since few data sets in the world even approach a petabyte in size, it is not at all clear how many MR users really need 1,000 nodes.”

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, data replicated 3x

DBMS-X: two phases

- Each node reads from the local file system *and redistributes tuples*
  - *LOAD command issued in parallel but actually executed sequentially*
- Reorganize data on each node (e.g., compress data, build index) *in parallel*

Vertica: Load data in parallel and automatically sorted and compressed

- Has a column-oriented storage

Data Loading

Data Inputs: (2 Data sets)

1. Weak scaling: fix the size of data per node (535MB/node), add nodes and data
2. Strong scaling: fix the total data size (1TB), add nodes

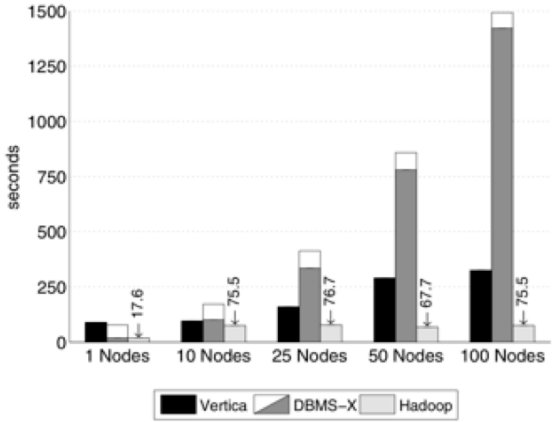

Figure 1: Load Times – Grep Task Data Set (535MB/node)

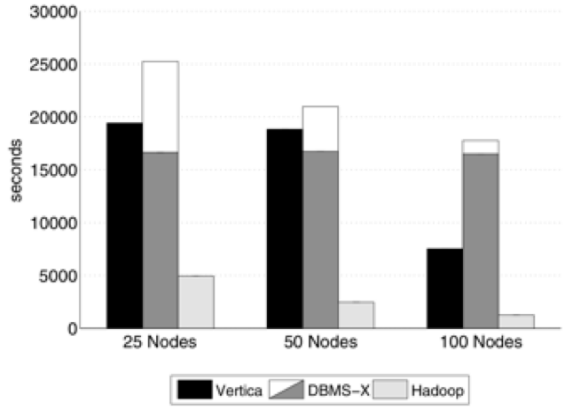

Figure 2: Load Times – Grep Task Data Set (1TB/cluster)

Hadoop outperforms parallel DBMS

Data Loading

MapReduce's Defence:

- For many comparisons in this paper, the time for loading data in parallel DBs is 5-50x the time needed to analyze the data via Hadoop.

Data Inputs: (2 Data sets)

1. Weak scaling: fix the size of data per node (535MB/node), add nodes and data
2. Strong scaling: fix the total data size (1TB), add nodes

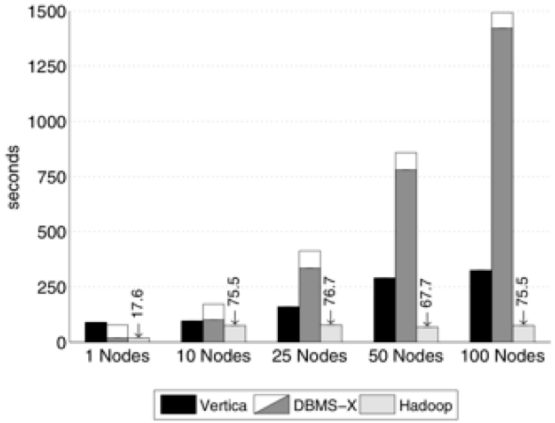

Figure 1: Load Times – Grep Task Data Set (535MB/node)

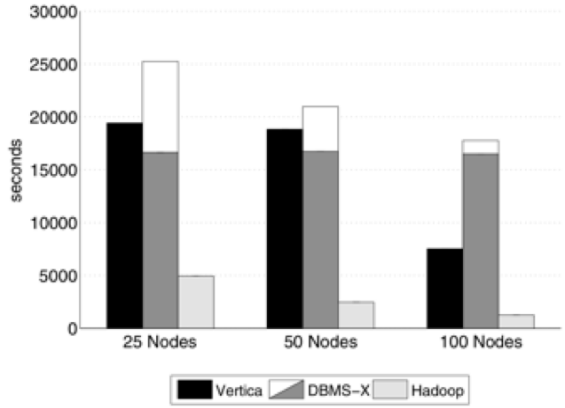

Figure 2: Load Times – Grep Task Data Set (1TB/cluster)

Hadoop outperforms parallel DBMS

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

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

MapReduce's Defence:

- This merging is unnecessary!

For each task, Hadoop needs to do an additional Reduce job to combine the output into a single file

Grep Task Execution Performance

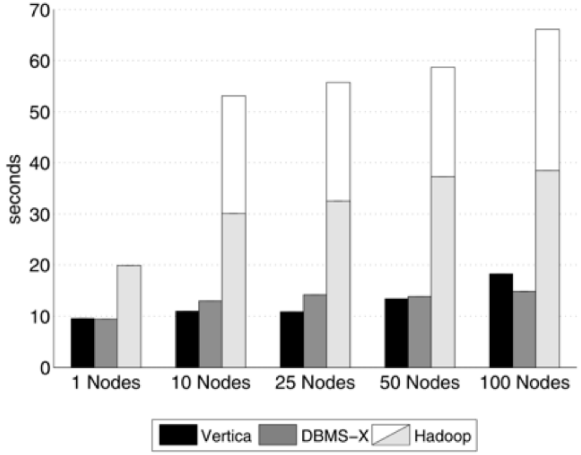

Figure 4: Grep Task Results – 535MB/node Data Set

(Fix the size of data per node)

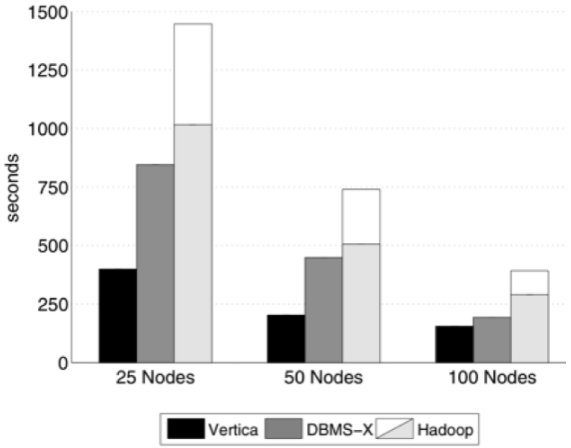

Figure 5: Grep Task Results – 1TB/cluster Data Set

(Fix the total data size)

Hadoop

- High start-up overhead
- Centralized job tracker (hurts scalability)

Vertica

- Fast due to aggressive use of data compression

Grep Task Execution Performance

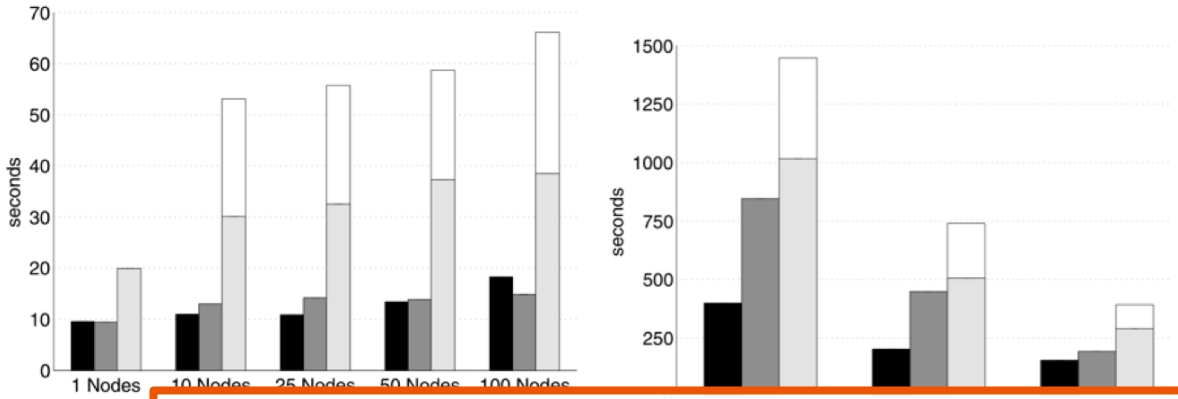

Figure 4: G ep Task Results – 535MB/node Data Set

(Fix th size

Hadoop

- High start-up overhead
- Centralized job tracker (hurts scalability)

Vertica

- Fast due to aggressive use of data compression

(Fix the total data size)

MapReduce's Defence:

- High start-up overhead is due to the immature implementation, not fundamental differences in programming models
- (Again) Merging (top segment for Hadoop) is unnecessary

Select & Join Performances

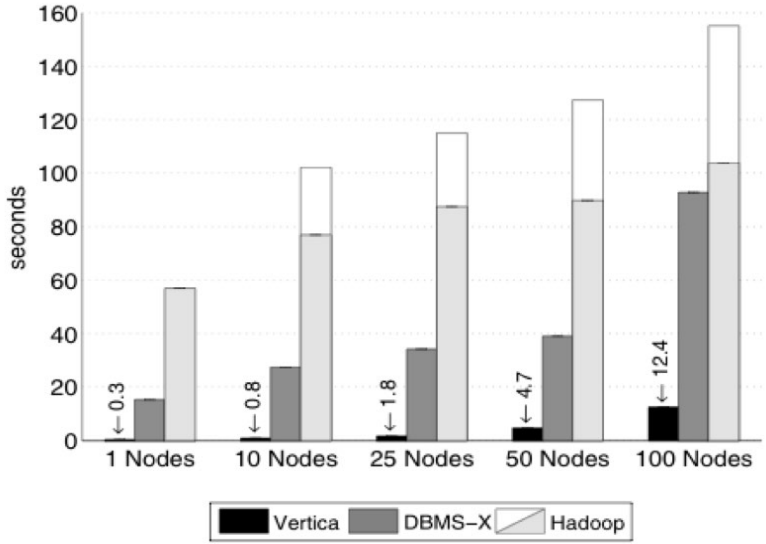

Figure 6: Selection Task Results

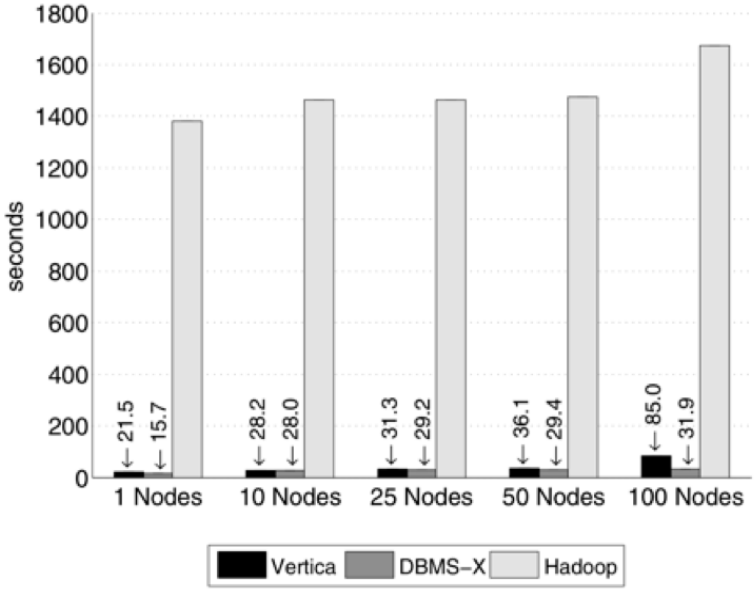

Figure 9: Join Task Results

Parallel DBs can use indexes so are much faster

Select & Join Performances

MapReduce's Defence:

- The conclusion is invalid, as it is based on a misconception that MapReduce cannot use indexes

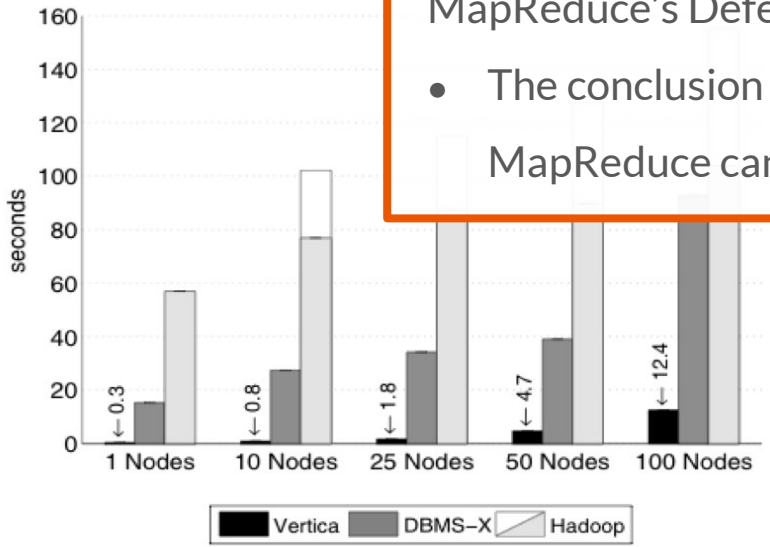

Figure 6: Selection Task Results

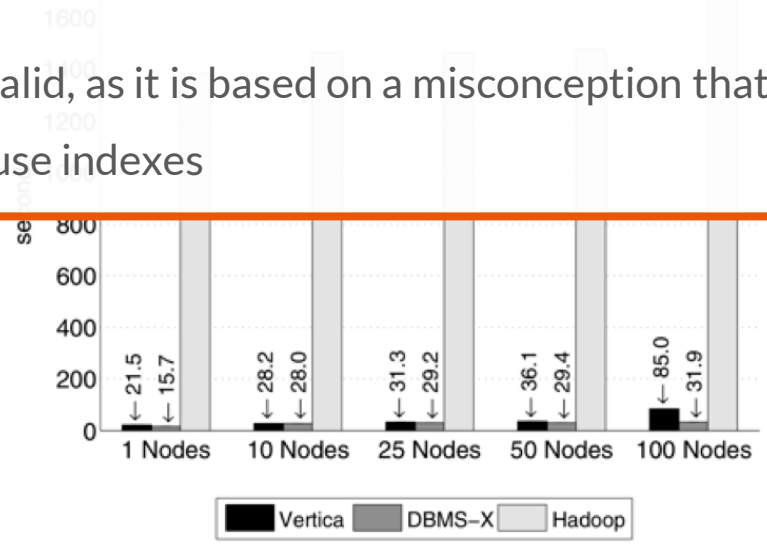

Figure 9: Join Task Results

Parallel DBs can use indexes so are much faster

Aggregation Performances

Two versions, both compute the sum of adRevenue in each group but have different # of groups

- Parallel DBs outperform Hadoop
- Vertica is the best because of its column store
- Runtimes are constant with the number of nodes, as they are bounded by the “constant sequential scan performance and network repartitioning costs for each node”

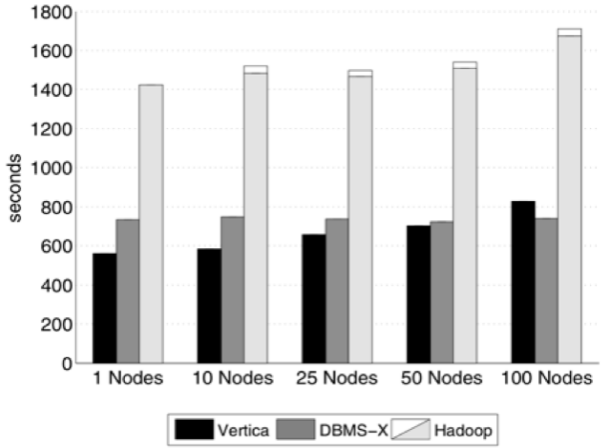

Figure 7: Aggregation Task Results (2.5 million Groups)

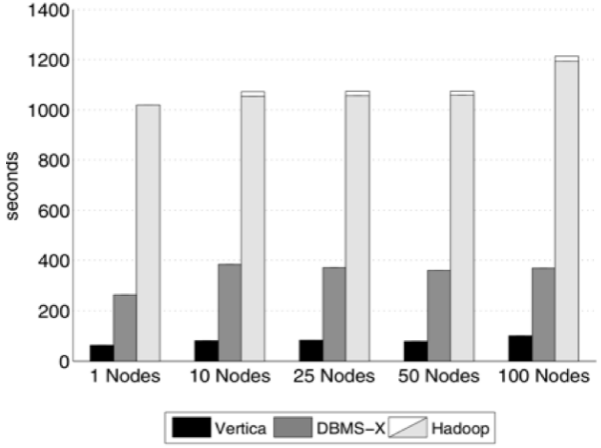

Figure 8: Aggregation Task Results (2,000 Groups)

UDF Aggregation Performances

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

- Hard to express in SQL (even with UDF)
- DBs require executing extra UDF/parser to load data (bottom segment of the stack)
- DBs perform worse than Hadoop due to UDF/parsing interacting with external data

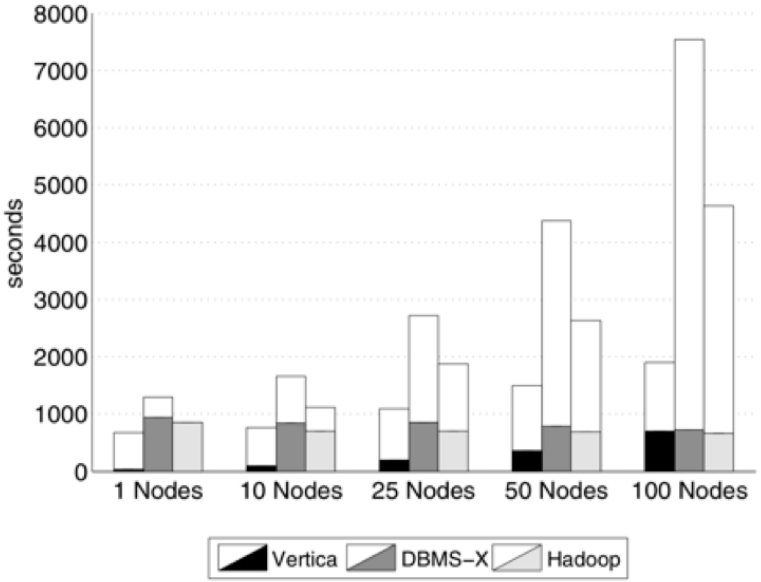

Figure 10: UDF Aggregation Task Results

UDF Aggregation Performances

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

- Hard to express in SQL (even with UDF)
- DBs require executing extra UDF/parser to load data (bottom segment of the stack)
- DBs perform worse than Hadoop due to UDF/parsing interacting with external data

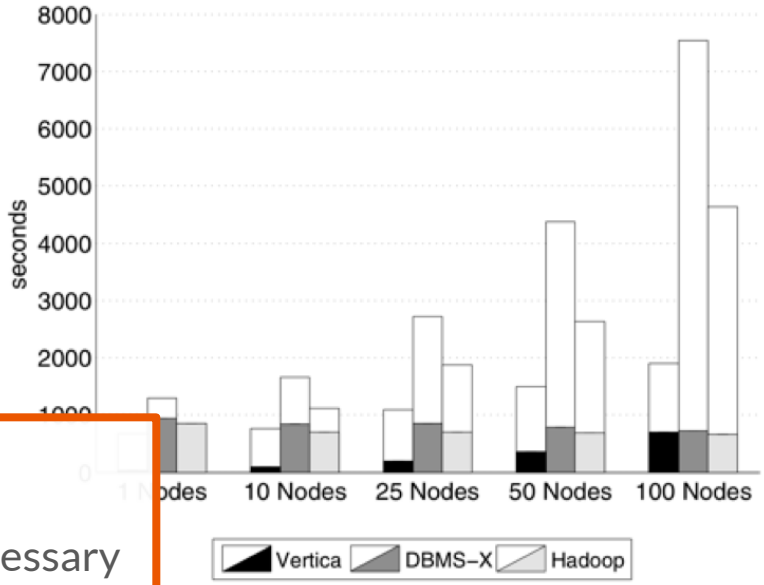

Figure 10: UDF Aggregation Task Results

MapReduce's Defence:

- (Again) Merging (top segment for Hadoop) is unnecessary

Discussion Question (Groups of 4, 1 systems person in each)

Properly benchmarking a system is a difficult, let alone comparing several systems that are built different.

- What are the factors you should consider to ensure a fair comparison of two systems?
- As the author of a paper, what information should you provide to ensure the reproducibility of your experiment results?

Summary: Why Are Parallel DBs Faster Than MapReduce?

- Can use indexes; don’t always requiring a full scan over input data
- Don’t have to stores input data in plain text files after loading it
- Can have novel storage mechanisms (e.g., column-orientation)
- Can use aggressive compression techniques
- Have sophisticated parallel algorithms for querying large amounts of data efficiently (query optimization)

Summary: Why Are Parallel DBs Faster Than MapReduce?

- Can use indexes; don't always requiring a full scan over input data
- Don't have to stores input data in plain text files after loading it
- Can have novel storage mechanisms (e.g., column-orientation)
- Can use aggressive compression techniques
- Have sophisticated parallel algorithms for querying large amounts of data efficiently (query optimization)

MapReduce's Defence:

- MR can use Protocol Buffer as a schema and for reading/writing data. **This can speed up record-parsing by 80x.**
- MR can also use input sources that have been indexed
- The step for merging results using a single reducer is unnecessary.
- Many issues are only implementation and evaluation shortcomings

Summary: Why Is MapReduce Better Than Parallel DBs?

- Heterogeneous system: supports a mix of storage systems (e.g., distributed file systems, database query results, BigTable)
  - MR provides a simple model for analyzing data in heterogeneous systems.
- Easy and fast loading: important as “data sets are often generated, processed once or twice, and then **discarded**”
  - *“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)
- Easy to setup and use (from the first paper)

Discussion Question (Groups of 3)

MapReduce misconceptions:

- Why are there “incorrect understandings” about 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 technology, how can you prove your point while maintaining a neutral stance?
- 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?

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)*

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
- Native support for JSON/BSON become desirable
- Not every programmer wants to learn SQL
- Stringent response time requirements
- Better scalability is needed for very large volumes of data
- Better fault tolerance is needed due to larger number of nodes
- More relaxed consistency requirements

Observed Problems of Current NoSQL DBs

- Not thinking about locking, storage management and recovery concurrently. Adding them later will be very hard (lessons from ARIES).
- Don’t scale in terms of concurrency (not supporting fine granularity of locking/latching).
- Lack of standards and complicated & varying data models .
- Forgot the benefits of high-level languages and data independence.
- Lack of general indexing.
- Implementation shortcuts that hurt scalability.
- Not supporting ACID is at least an oversimplification.

Discussion Question (Groups of 3)

“All the decades of evangelization that went on about the goodness of standards seems to have been forgotten in the context of NoSQL systems”

- The author claims that the lack of standards in NoSQL is going to be a nightmare in due course of time. Do you agree with this? Do you have any examples in your field that proved this wrong?
- When is a good point in time to talk about standardization?