

Data Management in the Cloud Lecture 6

MAP/REDUCE VERSUS DBMS

1

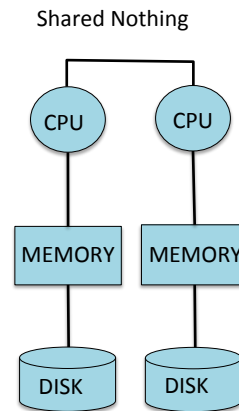
Map/Reduce Criticism

- Release of Map/Reduce caused a big reaction from the database community
 - The database community was initially very critical of Map Reduce
 - Now most DB people seem to believe that Map/Reduce style models and Parallel DBs will co-exist
- Initial arguments: “Why not use a parallel DBMS instead?”
 - map/reduce is a “giant step backwards”
 - no schema, no indexes, no high-level language
 - not novel at all (NCR Teradata)
 - does not provide features of traditional DBMS – indices, optimization, declarative query language
 - incompatible with DBMS tools

2

MapReduce - Comments

- Basic control flow for MapReduce has existed in parallel DBMS systems for years
- Almost any parallel processing task can be written as a set of database queries (UDFs/UDAs) or a set of MapReduce jobs
- Similarities
 - MR & P-DBMS both use “shared-nothing”
 - MR & P-DBMS both divide data into partitions / shards



3

Architectural Elements - Schema

DBMS	MapReduce
Schema Defined in Database	Schema defined in MR programs
Must define schema in advance (schemas are difficult!)	Easy to get started...
Schema is separate from application (re-use / sharing is easy)	Each MR program must parse the data and data structures in the MR files (sharing is difficult); programmers need to agree on structure

4

Architectural Elements – Indexing

PDBMS	MapReduce
Indices: increase load time, but greatly improve performance	No built-in indices: easy to get started, but performance may suffer
Indices maintained by database, can be used by any user	Programmer implement indices? Reuse?

5

Architectural Elements – Programming Model & Flexibility

DBMS	MapReduce
Programming Model: High-level / SQL	Programming Model: Lower-level (procedural specification) Widespread sharing of code fragments High-level languages added – Pig/Hive
Flexibility: MR proponents: “SQL does not facilitate the desired generality that MR provides,” but DBMSs have UDFs/UDAs	Flexibility: High flexibility - programming language...

Quote Credit: “A Comparison of Approaches to Large-Scale Data Analysis” by A. Pavlo et al., 2004

6

Architectural Elements – Execution Strategy & Fault Tolerance

DBMS	MapReduce
Disk Access: Database has coordinated, optimized disk access. Sends computation to disk.	Disk Access: 500,000 output files of Map, each Reducer pulls 1000 files → poor disk performance. Sends computation to disk only for initial Map reads.
Optimization: Sophisticated query optimization	Optimization: No automatic optimization. No selection push down.
Fault Tolerance: Avoid saving/writing intermediate work, restart larger granules	MR – more sophisticated fault-tolerance; better at handling node failures in the middle of computation (local materialization vs. streaming/push)

7

MapReduce – Performance Comments

- Performance experiments show tradeoffs
 - Parallel DBMSs require time to load & tune, but generally have shorter execution times
 - MapReduce generally has longer execution times

8

MR vs. PDBMS Performance Analysis

- Systems
 - parallel DBMS (Vertica and DBMS-X) vs. map/reduce (Hadoop)
- Tasks
 - original map/reduce task: “grep” from Google paper
 - typical database tasks: selection, aggregation, join, UDF
- Cluster
 - 100-node cluster
- Comments:
 - MR can scale to 1000’s of nodes, but may not be necessary with efficient parallel DBMSs
 - Few data sets are really petabyte size – not many users really need 1000 nodes

9

Performance - Setup

- 5 tasks (Grep, 4 Analytic)
- 3 systems (Hadoop, DBMS-X, Vertica)
- 100-node cluster, 2.4 GHz Intel Core 2 Duo, Red Hat Linux, 4GB RAM, two 250 GB SATA-I hard disks
- Experiments run on 1, 10, 25, 50 and 100 nodes
- Two Data Sets:
 - 535 MB/node : fixes amount of data per node (amount of data increases as # nodes increase)
 - 1TB total : fixes total amount of data (data per node decreases as # nodes increase)
 - Note: original MR paper had 1TB on 1800 nodes, 535 MB/node

10

Grep Task: Load

- Hadoop
 - Data loaded as plain text using command-line utility
 - No need for custom data loader
 - DBMS-X
 - Load command executed in parallel
 - Redistribute tuples to other node based on partitioning attribute
 - Reorganize on each node (compress, indices, housekeeping)
 - Vertica
 - Similar to DBMS-X
- SQL: `SELECT * FROM Data WHERE field like '%XYZ';`

11

Grep Task: Load Times

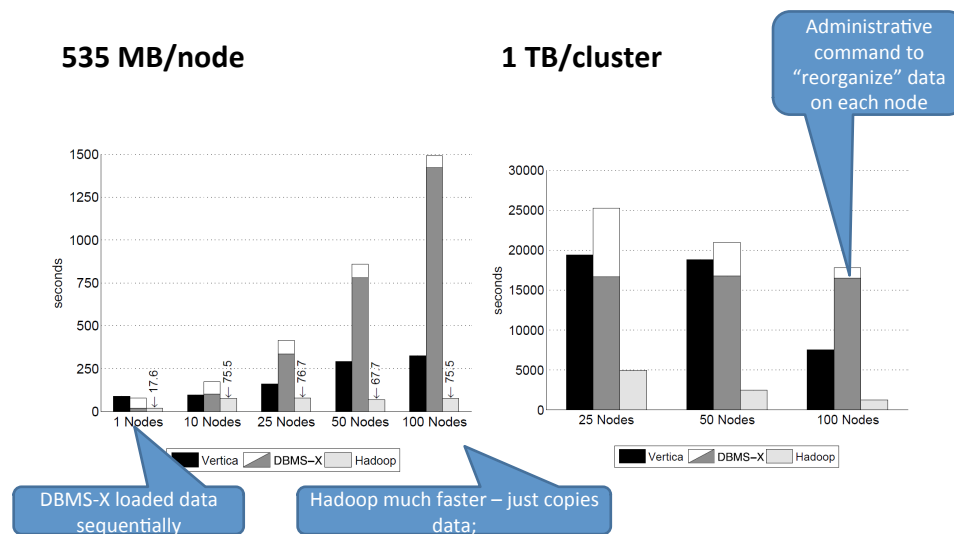


Figure Credit: "A Comparison of Approaches to Large-Scale Data Analysis" by A. Pavlo et al., 2004

12

Grep Task: Execution Times

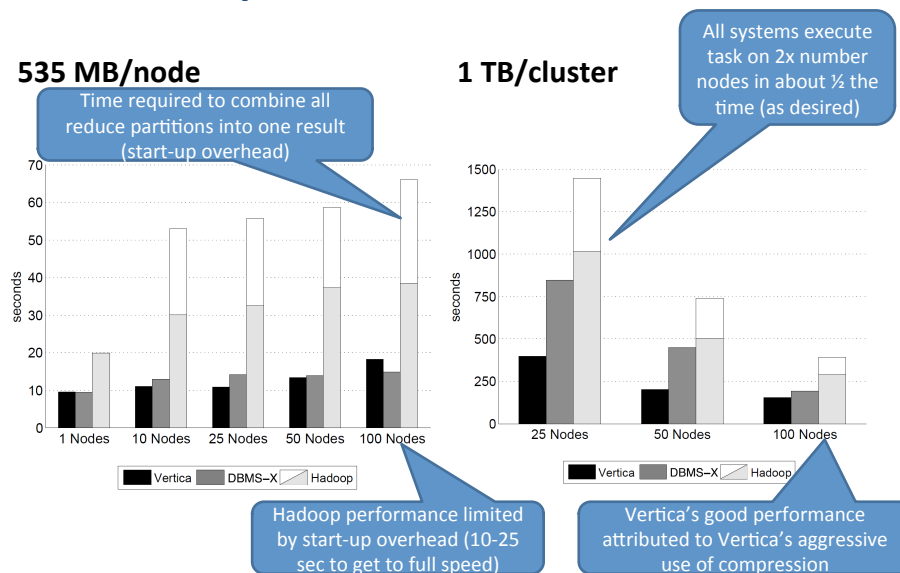


Figure Credit: "A Comparison of Approaches to Large-Scale Data Analysis" by A. Pavlo et al., 2004

13

Analytical Tasks

```
CREATE TABLE Documents (
  url VARCHAR(100)
    PRIMARY KEY,
  contents TEXT );

CREATE TABLE Rankings (
  pageURL VARCHAR(100)
    PRIMARY KEY,
  pageRank INT,
  avgDuration INT );

CREATE TABLE UserVisits (
  sourceIP VARCHAR(16),
  destURL VARCHAR(100),
  visitDate DATE,
  adRevenue FLOAT,
  userAgent VARCHAR(64),
  countryCode VARCHAR(3),
  languageCode VARCHAR(3),
  searchWord VARCHAR(32),
  duration INT );
```

- Data set (generated)
 - 600K unique HTML documents, with unique URL
 - Links to other pages randomly generated
 - 155M user visit records (20 GB/node)
 - 18M ranking records (1 GB/node)
- Loading
 - DBMS-X and Vertica use a UDF to process documents (temp table) → no load results given
 - Map-Reduce – load time decreased by 3 due to custom data loader (but no custom input handler)

14

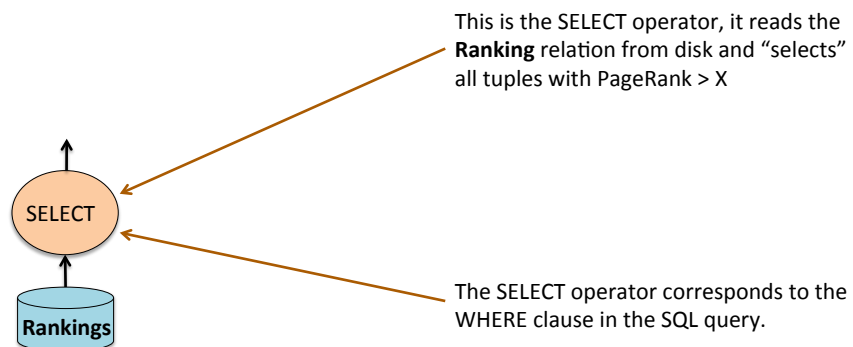
Selection

- SQL: **SELECT** pageURL, pageRank **FROM** Rankings
WHERE pageRank > X
- Map Function: Splits input value based on delimiter, outputs pageURL and pageRank if pageRank > X
- Reduce Function: none/identity

15

Database Execution - Selection

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

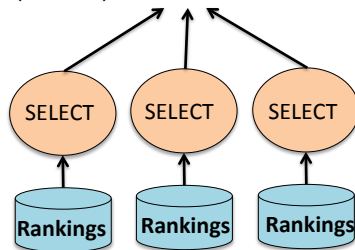


16

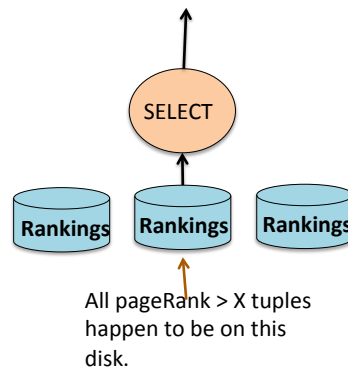
Parallel Database Execution - Selection

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

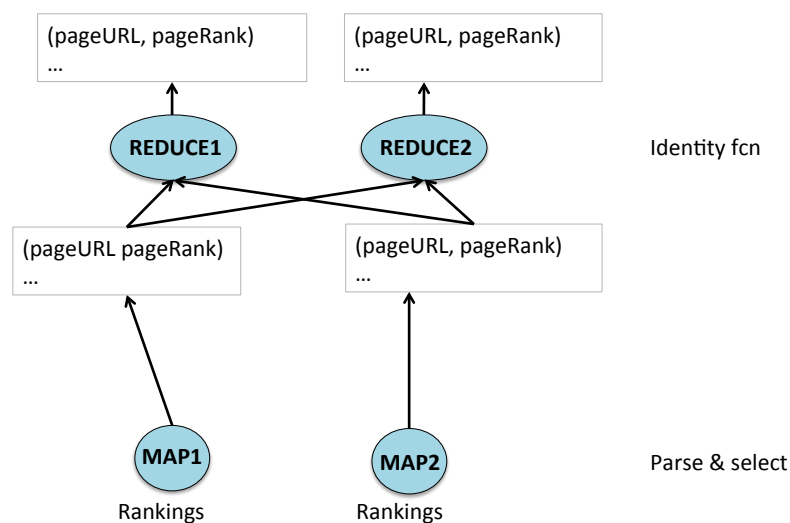
Case 1: Tuples from **Rankings** are randomly or hash partitioned (sharded) across the three disks.



Case 2: Tuples from **Ranking** are partitioned (sharded) based on pageRank.



Selection – Map Reduce



18

Selection Task

- SQL Query

```
SELECT pageURL, pageRank
FROM Rankings
WHERE pageRank > x
```

- Relational DBMS use index on pageRank column
- Relative performance degrades as number of nodes and amount of data increases
- Hadoop start-up cost increase with cluster size

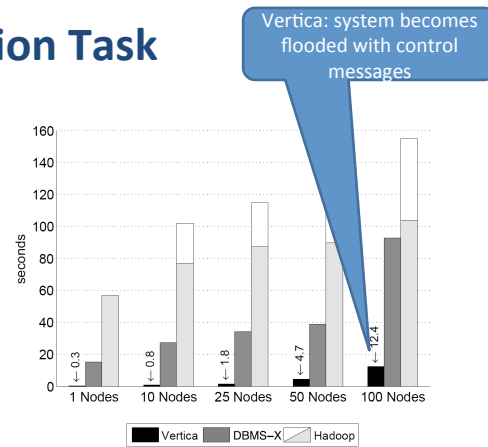


Figure Credit: "A Comparison of Approaches to Large-Scale Data Analysis" by A. Pavlo et al., 2004

19

Aggregation Task

- Calculate the total ad revenue for each source IP using the user visits table
- Task: performance of parallel analytics on a single read-only table where nodes need to exchange data to compute result
- DBMS execution: local group by, groups merged at coordinator

- Variant 1:** 2.5M groups

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

- Variant 2:** 2,000 groups

```
SELECT SUBSTR(sourceIP, 1, 7), SUM(adRevenue)
FROM UserVisits
GROUP BY SUBSTR(sourceIP, 1, 7)
```

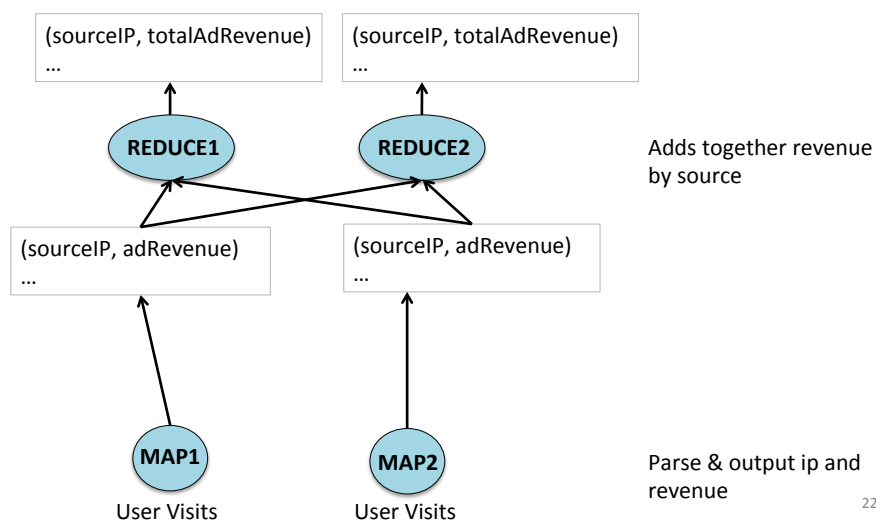
20

Aggregation

- SQL: `SELECT sourceIP, SUM(adRevenue) FROM UserVisits GROUP BY sourceIP;`
- Map Function: split by delimiter, outputs (sourceIP, adRevenue)
- Reduce Function: adds revenue for each sourceIP (uses a combiner)

21

Aggregation – Map Reduce



Parallel Database Execution - Sum

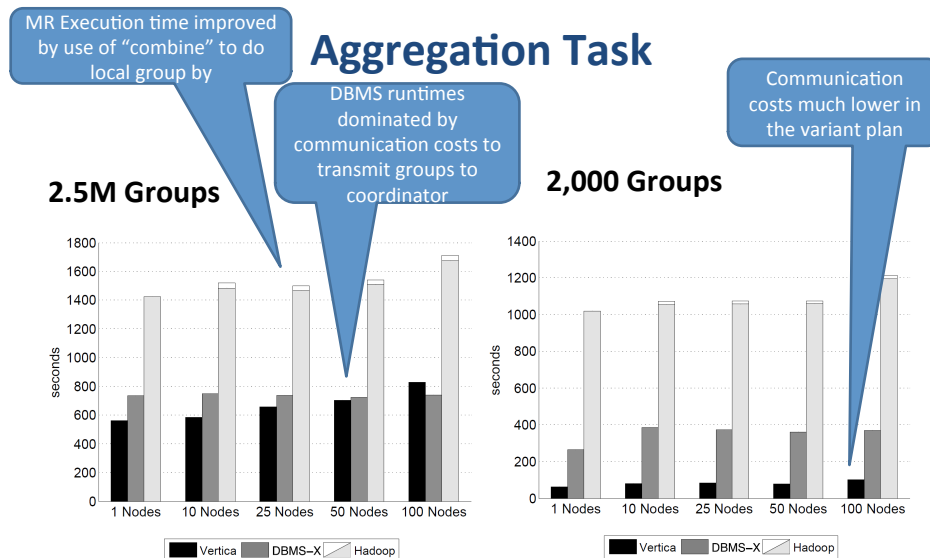
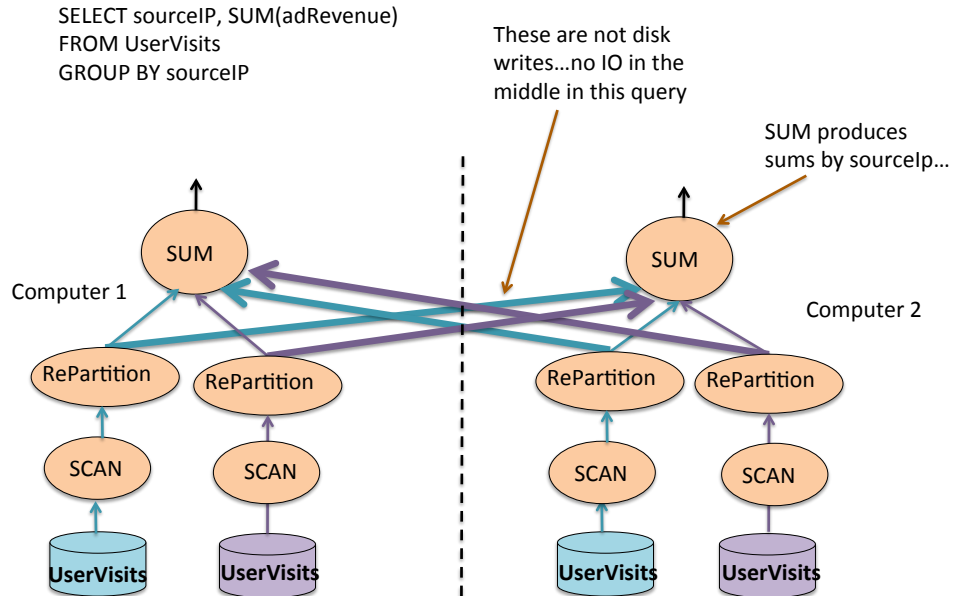


Figure Credit: "A Comparison of Approaches to Large-Scale Data Analysis" by A. Pavlo et al., 2004

Join Task

SQL Query

```
SELECT INTO Temp
  UV.sourceIP,
  AVG(R.pageRank) AS avgPageRank,
  SUM(UV.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,
  avgPageRank,
  totalRevenue
FROM Temp
ORDER BY totalRevenue DESC LIMIT 1
```

Map/reduce program

- Uses three phases
 - **Phase 1:** filters records outside date range and joins with rankings file
 - **Phase 2:** computes total ad revenue and average page rank based on source IP
 - **Phase 3:** produces the record with the largest total ad revenue
- Phases run in strict sequential order

In words: Find Url with highest total revenue and it's page rank

25

Join in MR

- Phase 1: filters records outside data range and joins with Rankings file
 - Input is all UserVisits and Rankings data files
 - Map: determine record type by counting number of fields
 - If UserVisits, apply date range predicate
 - Output – composite keys (destUrl, K1), (pageUrl, K2)
 - Hash function only on url portion of the key
 - Reduce
 - Input – single sorted run of records in URL order – divide into 2 sets and do cross product
- Phase 2: compute total adRevenue and average pageRank
 - Map: identity map fcn
 - Reduce gathers all records for a particular sourceIp on a single node
 - Reduce: computes adRevenue, pageRank – keep one with max total adRevenue

26

Join in MR

- Phase 3: find the record with the largest total adRevenue
 - Map: identity
 - Reduce: one reduce function to keep track of the record with the largest totalRevenue field

27

Database Execution - Join

Schema:

shoes (id integer, brand text, description text, size float, color text, lastworn date)

shoestorage (id integer, shelfnumber integer, shelfposition integer)

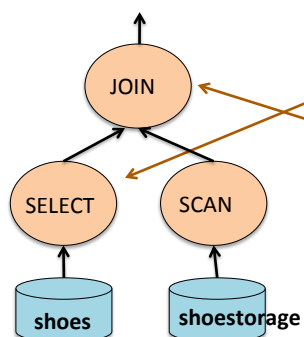
SELECT brand, description, size, shelfnumber, shelfposition

FROM **shoes**, **shoestorage**

WHERE shoes.id = shoestorage.id

AND color = 'Green'

AND lastworn < '1-25-2014'

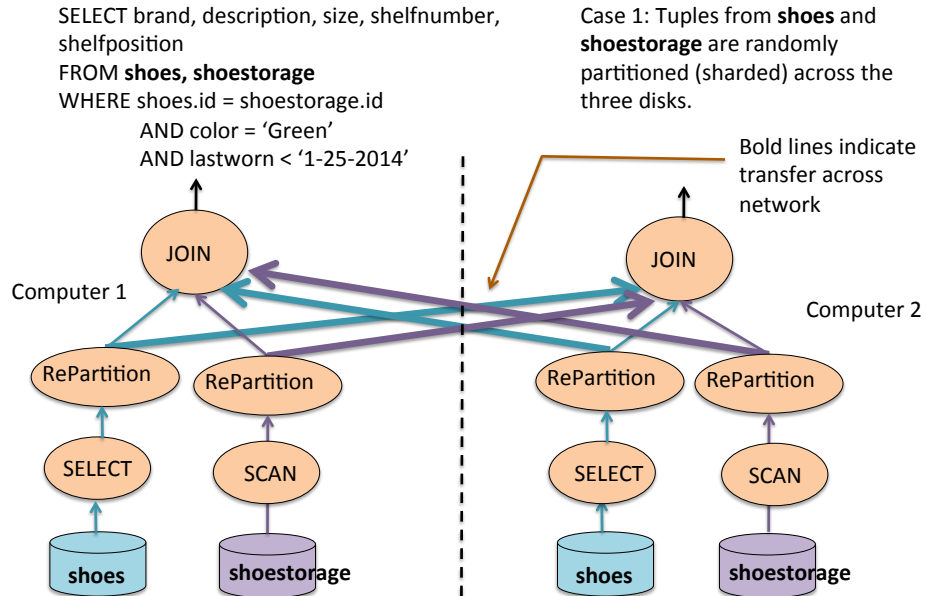


The SELECT operator “selects” all tuples containing green shoes that were last worn before 1-25-2014.

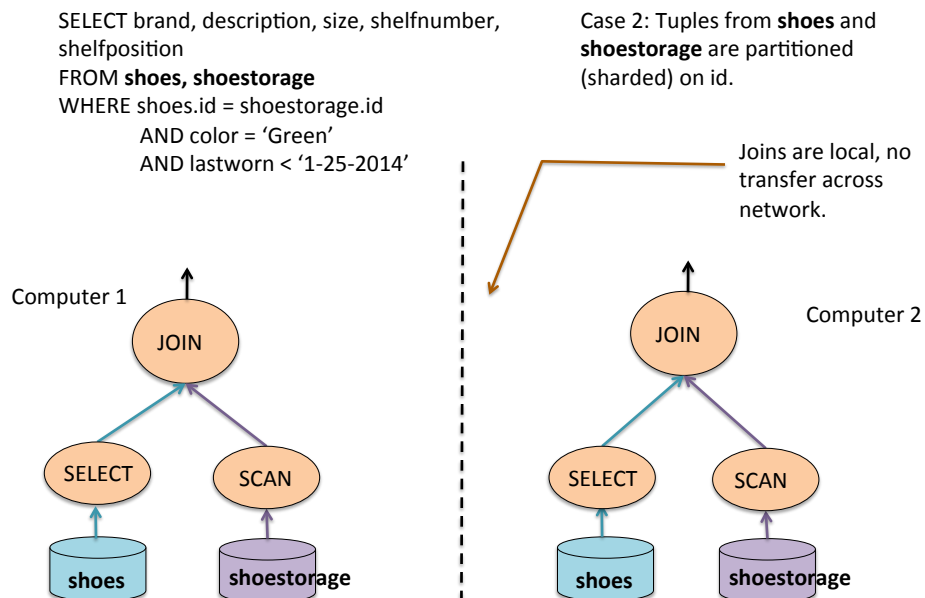
The JOIN operator combines the selected tuples from the **shoes** relation and the **shoestorage** to produce storage locations for the green shoes last worn before 1-25-2014.

28

Parallel Database Execution - Join



Parallel Database Execution - Join



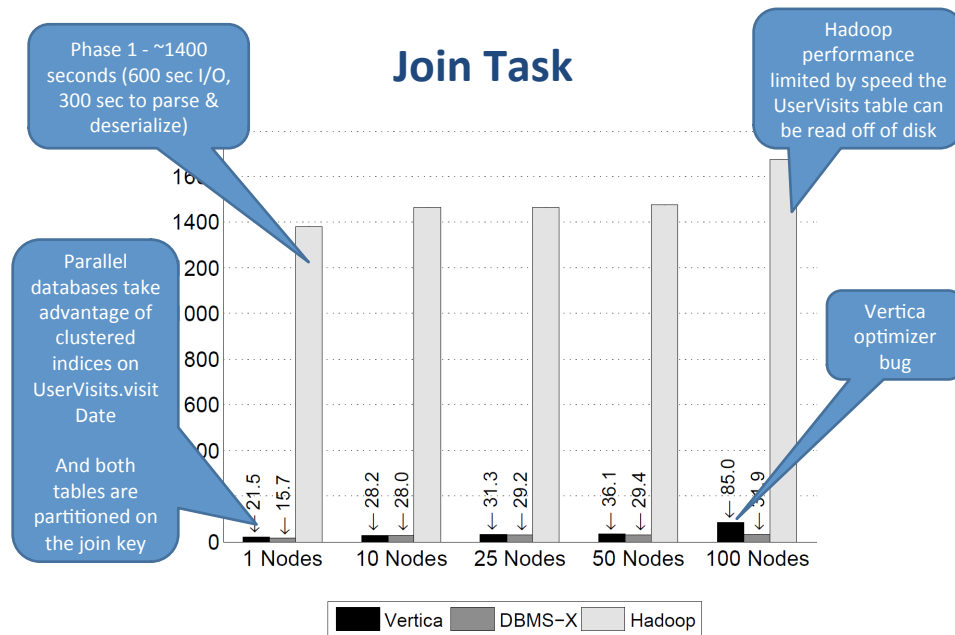


Figure Credit: "A Comparison of Approaches to Large-Scale Data Analysis" by A. Pavlo et al., 2004

31

UDF Aggregation Task

- Compute in-link count for each document in the data set
- SQL Query


```
SELECT INTO Temp UDF(contents) FROM Documents
SELECT url, SUM(value) FROM Temp GROUP BY url
```
- Map/reduce program
 - documents are split into lines
 - input key/value pairs: <line number, line contents>
 - **map**: uses regex to find URLs and emits <URL, 1> for each URL
 - **reduce**: counts the number of values for a given key
- DBMS
 - Requires UDF to parse contents of records in Document table – nearly identical to Map function (difficult to implement in DBMS)
 - DBMS-X: not possible to run UDF over contents stored as BLOB in database; instead UDF has to access local file system
 - Vertica: does not currently support UDF, uses a special pre-processor – processed file, write to disk, then loads...

32

UDF Aggregation Task

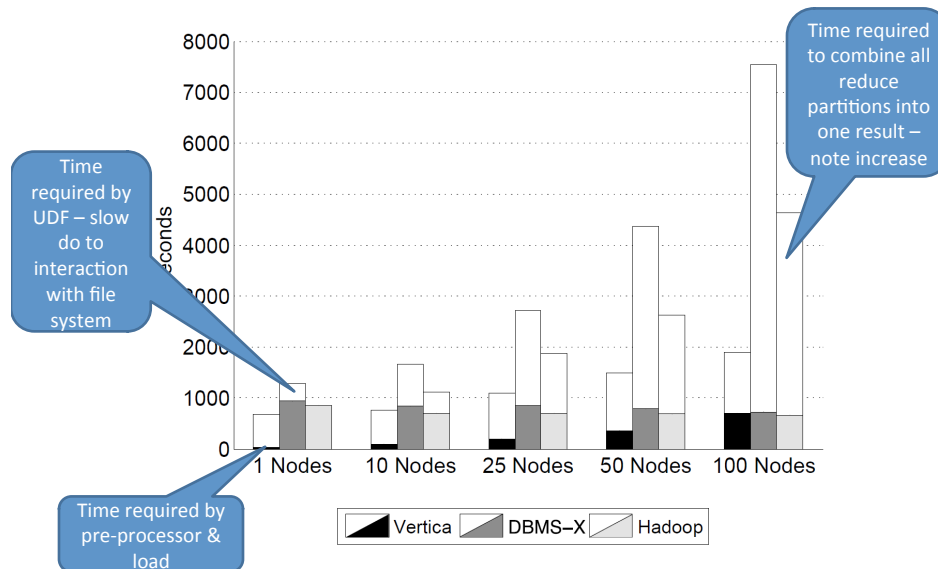


Figure Credit: "A Comparison of Approaches to Large-Scale Data Analysis" by A. Pavlo et al., 2004

33

Map/Reduce vs. Parallel DBMS

- No schema, no index, no high-level language
 - faster loading vs. faster execution
 - easier prototyping vs. easier maintenance
- Fault tolerance
 - restart of single worker vs. restart of transaction
- Installation and tool support
 - easy to setup map/reduce vs. challenging to configure parallel DBMS
 - no tools for tuning vs. tools for automatic performance tuning
- Performance per node
 - results seem to indicate that parallel DBMS achieve the same performance as map/reduce in smaller clusters

34

Discussion Question

1. What MapReduce feature would be useful in a DBMS?
2. What DBMS feature would be useful in MapReduce?

35

References

- A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker: **A Comparison of Approaches to Large-Scale Data Analysis**. *Proc. Intl. Conf. on Management of Data (SIGMOD)*, pp. 165-178, 2009.

36