Tradeoffs Between Parallel Database Systems, Hadoop, and HadoopDB as Platforms for Petabyte-Scale Analysis

Daniel Abadi
Yale University
November 23rd, 2011

Data, Data, Everywhere

- Data explosion
  - Web 2.0 → more user data
  - More devices that sense data
  - More equipment that produce data at extraordinary rates (e.g. high throughput sequencing)
  - More interactions being tracked (e.g. clickstream data)
  - More business processes are being digitized
  - More history being kept
- Data becoming core to decision making, operational activites, and scientific process
  - Want raw data (not aggregated version)
  - Want to run complex, ad-hoc analytics (in addition to reporting)

System Design for the Data Deluge

- Shared-memory does not scale nearly well enough for petascale analytics
- Shared-disk is adequate for many applications, especially for CPU intensive applications, but can have scalability problems for data I/O intensive workloads
- For scan performance, nothing beats putting CPUs next to the disks
  - Partition data across CPUs/disks
  - Shared-nothing designs increasingly being used for petascale analytics

Parallel Database Systems

- Shared-nothing implementations existed since the 80’s
  - Plenty of commercial options (Teradata, Microsoft PDW, IBM Netezza, HP Vertica, EMC Greenplum, Aster Data, many more)
  - SQL interface, with UDF support
  - Excels at managing and processing structured, relational data
  - Query execution via relational operator pipelines (select, project, join, group by, etc)

MapReduce

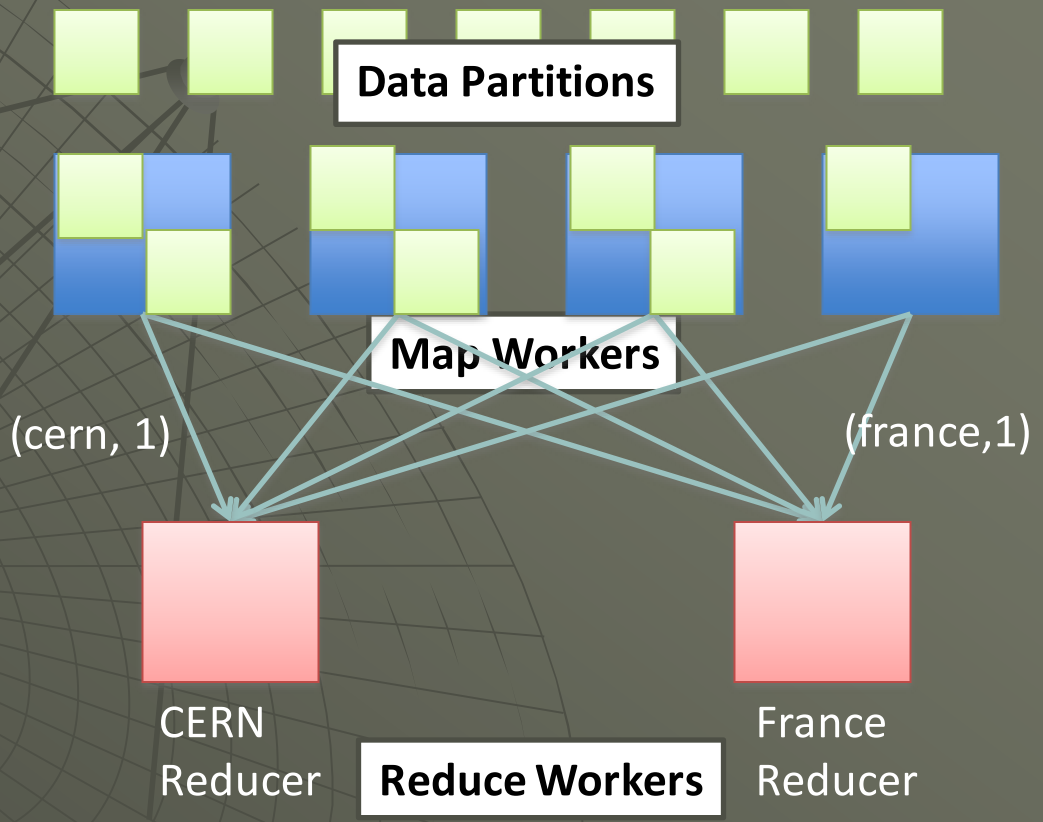
- Data is partitioned across N machines
  - Typically stored in a distributed file system (GFS/HDFS)
- On each machine n, apply a function, Map, to each data item d
  - Map(d) → {(key_1,$value_1$)}   "map job"
  - Sort output of all map jobs on n by key
  - Send (key_1,$value_1$) pairs with same key value to same machine (using e.g., hashing)
- On each machine m, apply reduce function to (key_1, {$value_1$}) pairs mapped to it
  - Reduce(key_1,{$value_1$}) → (key_2,$value_2$) "reduce job"
- Optionally, collect output and present to user

Example

- Count occurrences of the word "cern" and "france" in all documents

```
map(d):
    words = split(d,' ')
    foreach w in words:
     if w == 'cern'
              emit ('cern', 1)
     if w == 'france'
              emit ('france', 1)

reduce(key, valueSet):
    count = 0
    for each v in valueSet:
     count += v
    emit (key,count)
```



Relational Operators In MR

- Straightforward to implement relational operators in MapReduce
  - Select: simple filter in Map Phase
  - Project: project function in Map Phase
  - Join: Map produces tuples with join key as key; Reduce performs the join
- Query plans can be implemented as a sequence of MapReduce jobs (e.g Hive)

Overview of Talk

- Compare these two approaches to petascale data analysis
- Discuss a hybrid approach called HadoopDB

Similarities

- Both are suitable for large-scale data processing
  - I.e. analytical processing workloads
  - Bulk loads
  - Not optimized for transactional workloads
  - Queries over large amounts of data
  - Both can handle both relational and nonrelational queries (DBMS via UDFs)

Differences

- MapReduce can operate on *in-situ* data, without requiring transformation or loading
- Schemas:
  - MapReduce doesn't require them, DBMSs do
  - Easy to write simple MR programs
- Indexes
  - MR provides no built in support
- Declarative vs imperative programming
- MapReduce uses a run-time scheduler for fine-grained load balancing
- MapReduce checkpoints intermediate results for fault tolerance

Key (Not Fundamental) Difference

- Hadoop
  - Open source implementation of MapReduce
- There exists no widely used open source parallel database system
  - Commercial systems charge by the Terabyte or CPU
  - Big problem for "big data" companies like Facebook

Goal of Rest of Talk

- Discuss our experience working with these systems
  - Tradeoffs
  - Include overview of SIGMOD 2009 benchmark paper
- Discuss a hybrid system we built at Yale (HadoopDB)
  - VLDB 2009 paper plus quick overviews of two 2011 papers

Three Benchmarks

- Stonebraker Web analytics benchmark (SIGMOD 2009 paper)
- TPC-H
- LUBM

Web Analytics Benchmark

- Goals
  - Understand differences in load and query time for some common data processing tasks
  - Choose representative set of tasks that:
    - Both should excel at
    - MapReduce should excel at
    - Databases should excel at

Hardware Setup

- 100 node cluster
- Each node
  - 2.4 GHz Code 2 Duo Processors
  - 4 GB RAM
  - 2 250 GB SATA HDs (74 MB/Sec sequential I/O)
- Dual GigE switches, each with 50 nodes
  - 128 Gbit/sec fabric
- Connected by a 64 Gbit/sec ring

Benchmarked Software

- Compare:
  - Popular commercial row-store parallel database system
  - Vertica (commercial column-store parallel database system)
  - Hadoop

Grep

- Used in original MapReduce paper
- Look for a 3 character pattern in 90 byte field of 100 byte records with schema:

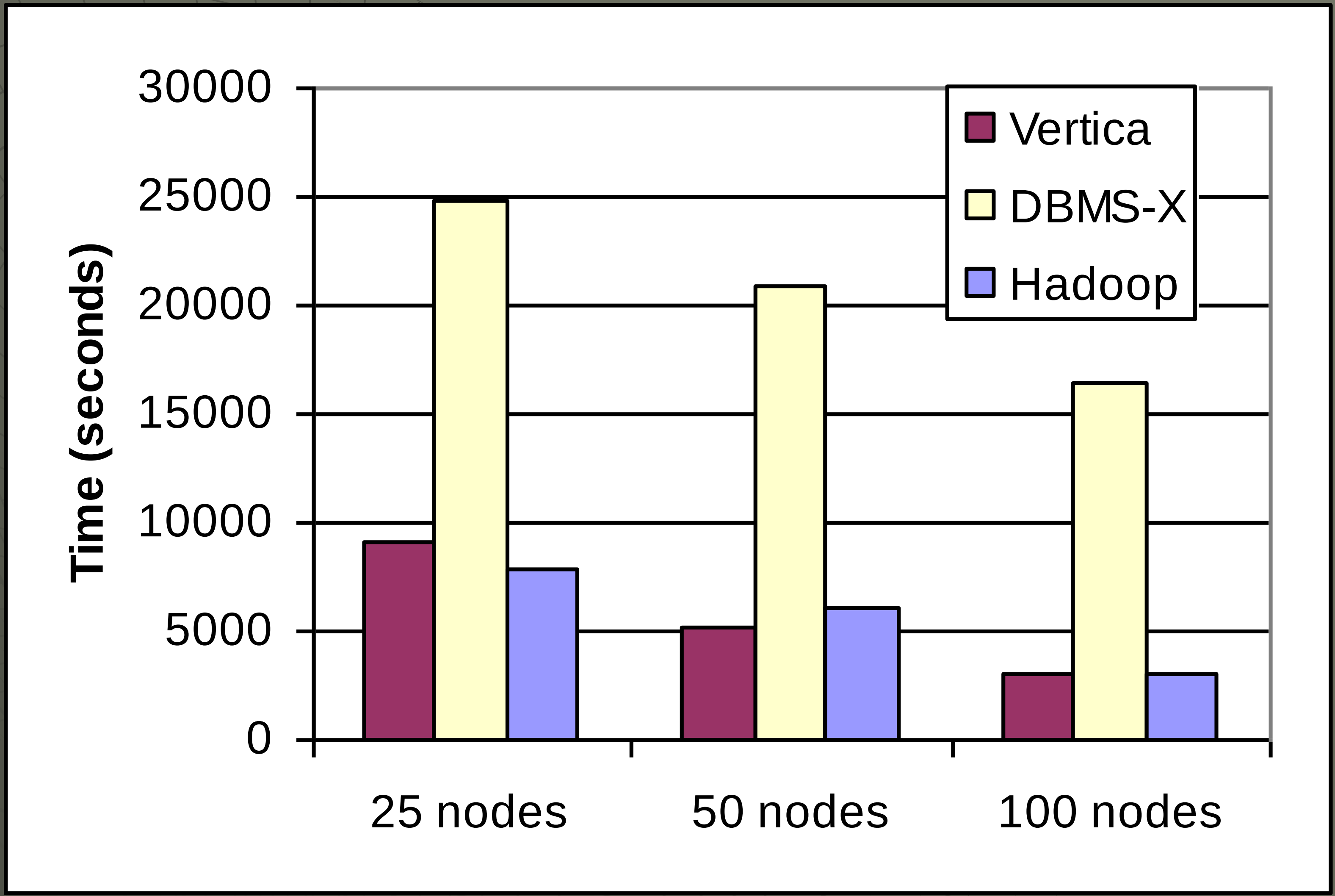
```
key VARCHAR(10) PRIMARY KEY
field VARCHAR(90)
```

  - Pattern occurs in .01% of records

```
SELECT * FROM T WHERE field LIKE '%XYZ%'
```

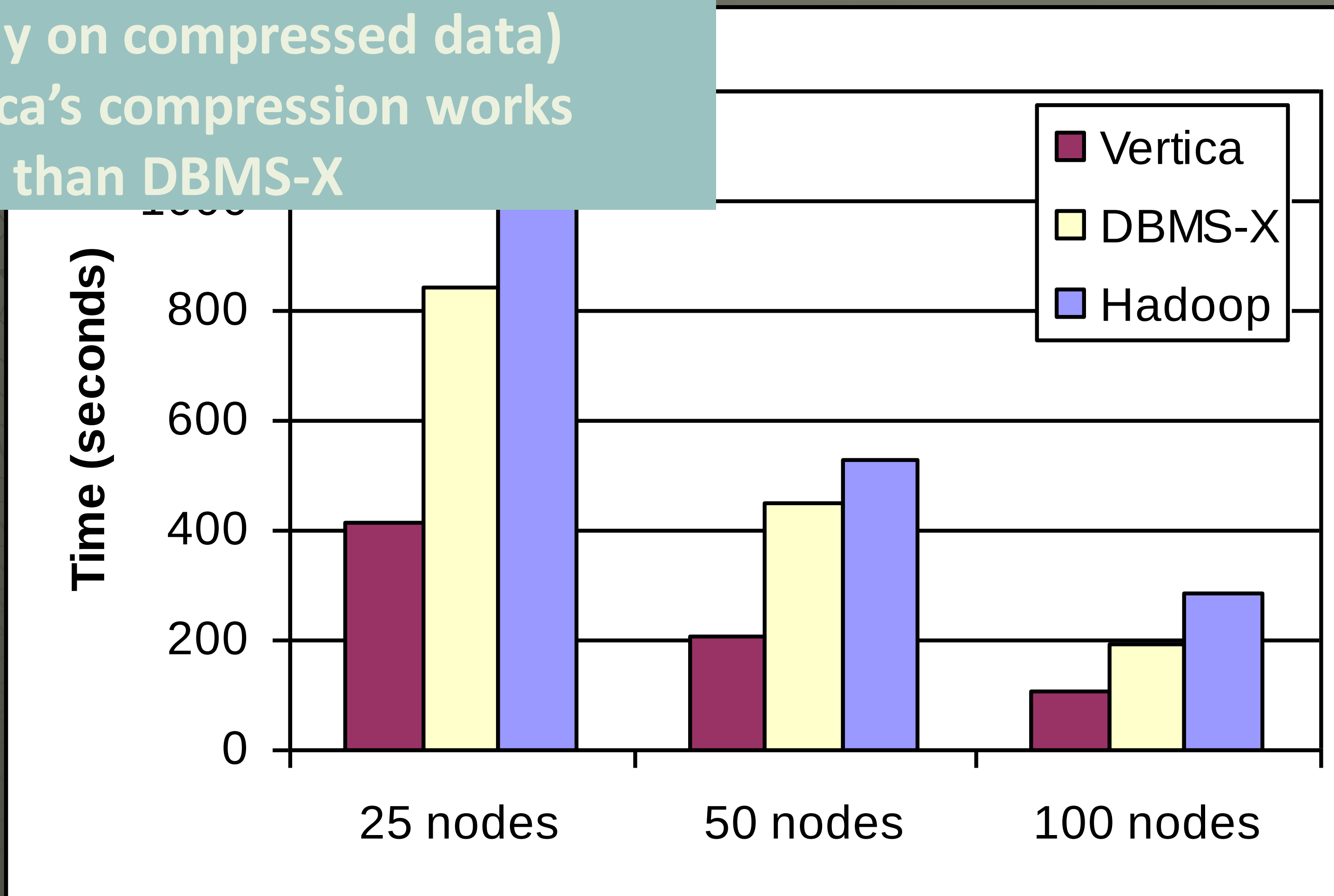
- 1 TB of data spread across 25, 50, or 100 nodes
  - ~10 billion records, 10–40 GB / node
- Expected Hadoop to perform well

1 TB Grep – Load Times



- All systems scale linearly (to 100 nodes)
- Database systems have better compression (and can operate directly on compressed data)
- Vertica’s compression works better than DBMS-X

uery Times



Analytical Tasks

- Simple web processing schema
- Task mix both relational and non-relational
- 600,000 randomly generated documents /node
  - Embedded URLs reference documents on other nodes
- 155 million user visits / node
  - ~20 GB / node
- 18 million rankings / node
  - ~1 GB / node

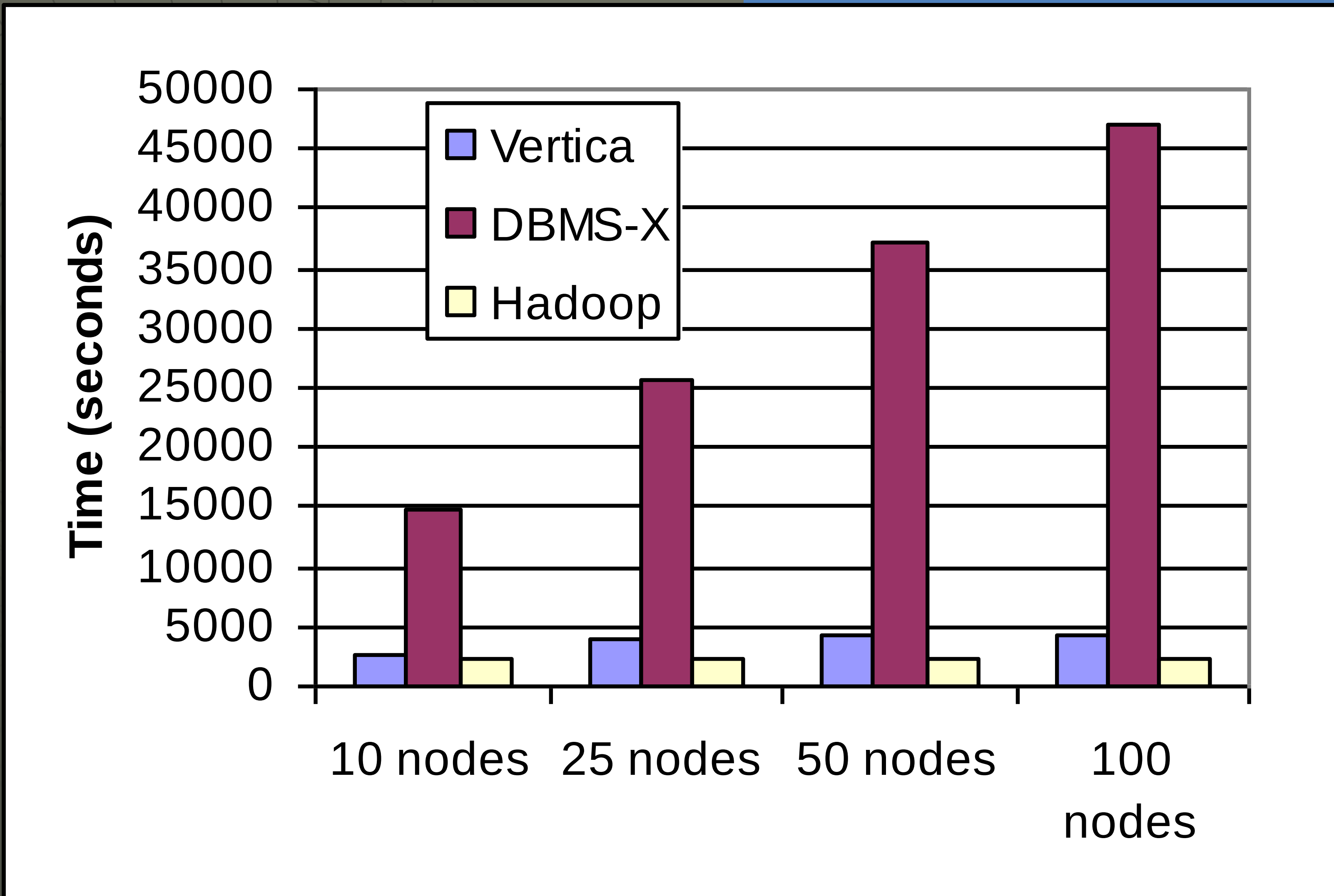
```
CREATE TABLE Documents (
    url VARCHAR(100) PRIMARY KEY,
    contents TEXT    );

CREATE TABLE UserVisits (
   sourceIP VARCHAR(16),
   destURL VARCHAR(100),
   visitDate DATE, adRevenue FLOAT,
   userAgent VARCHAR(64),
   countryCode VARCHAR(3),
   languageCode VARCHAR(6),
   searchWord VARCHAR(32),
   duration INT   );

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

Loading – User Visits

Other tables show similar trends



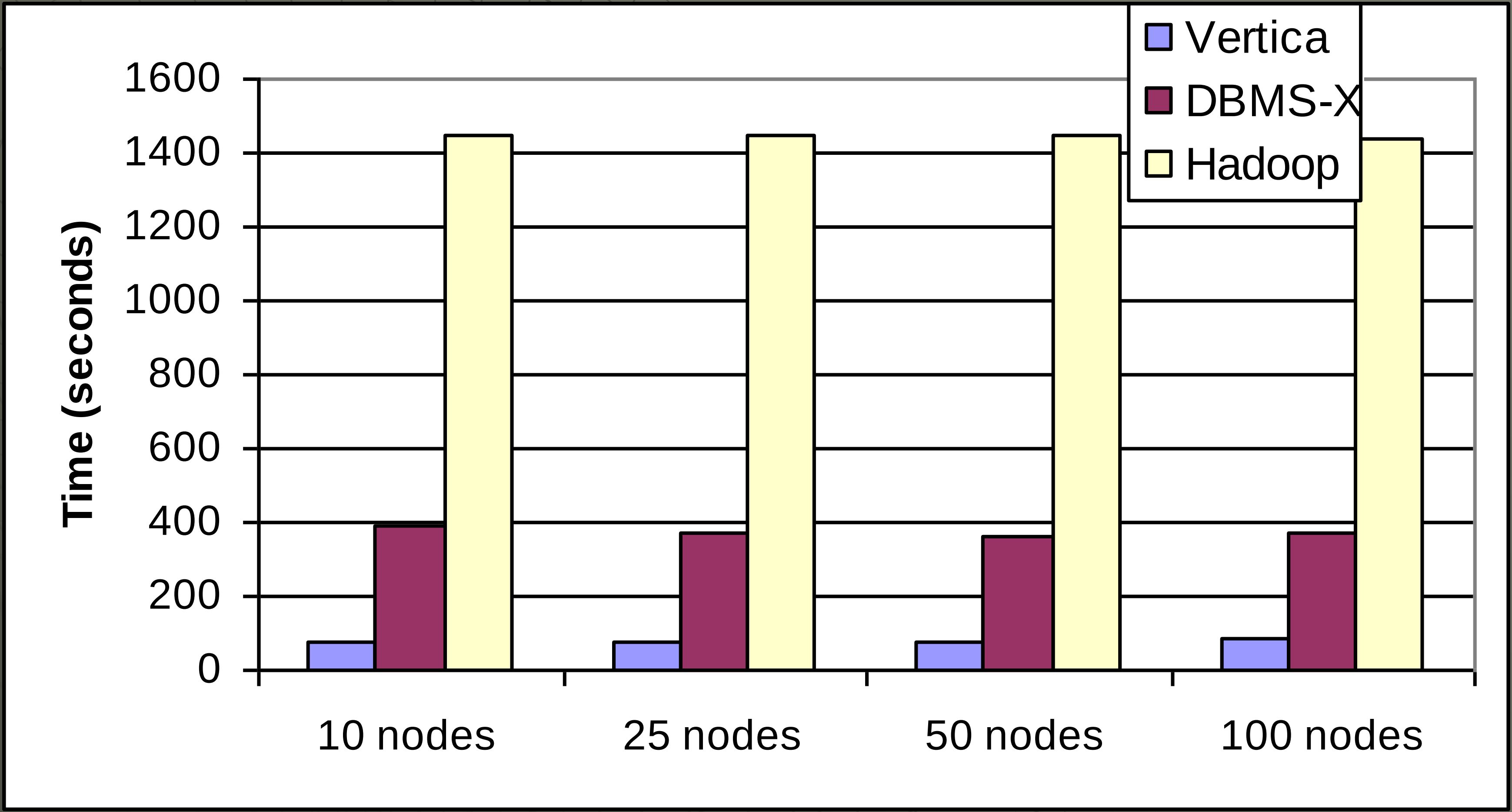
Aggregation Task

- Simple aggregation query to find adRevenue by IP prefix

```
SELECT SUBSTR(sourceIP, 1, 7), sum(adRevenue)
FROM userVistits GROUP BY SUBSTR(sourceIP, 1, 7)
```

- Parallel analytics query for DBMS
  - (Compute partial aggregate on each node, merge answers to produce result)
  - Yields 2,000 records (24 KB)

Aggregation Task Performance



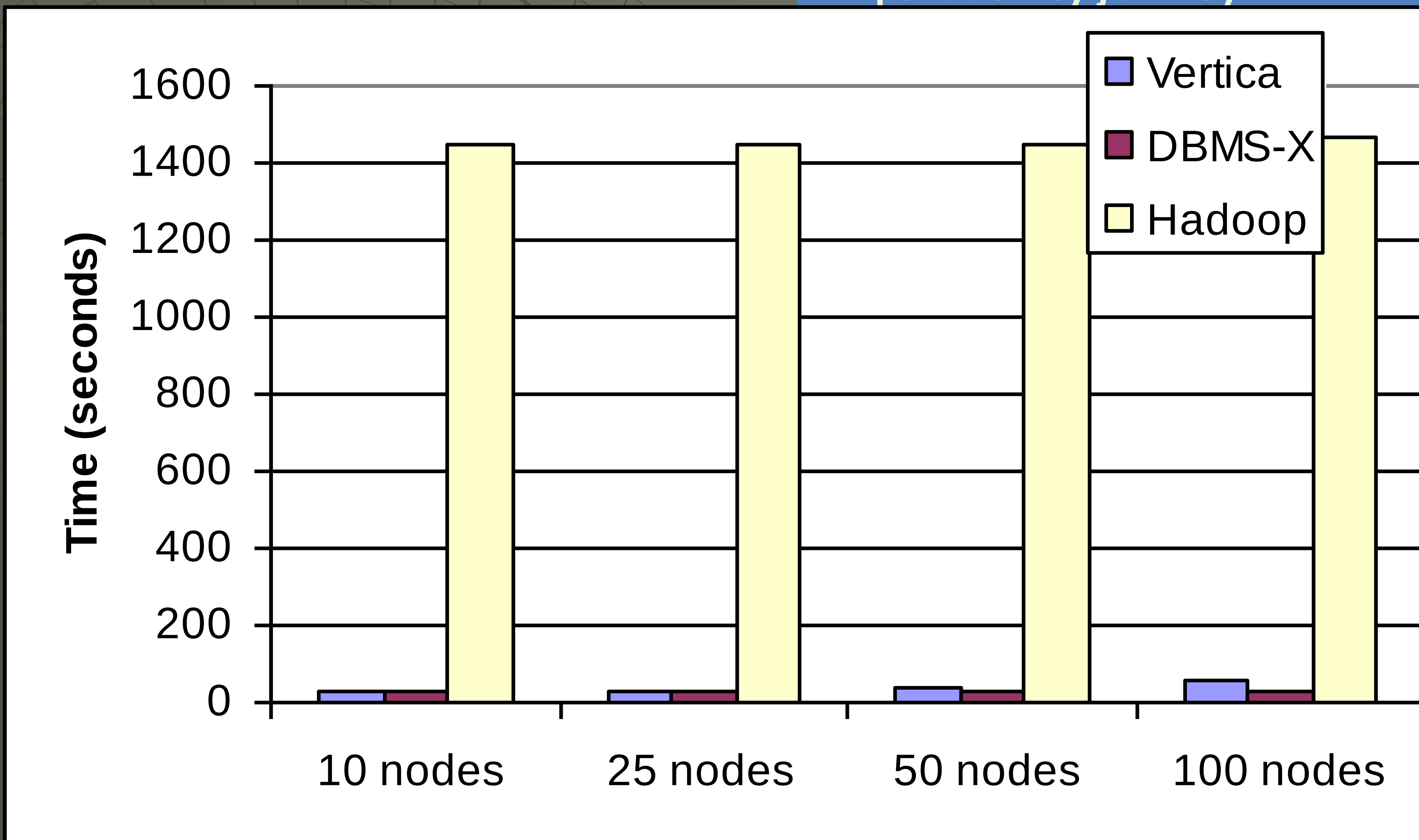
Join Task

- Join rankins and userVisits for sourceIP analysis and revenue attribution

```
SELECT sourceIP, AVG(pageRank), SUM(adRevenue)
FROM rankings, userVistits
WHERE pageURL=destURL
AND visitData BETWEEN 2000-1-15 AND 2000-1-22
GROUP BY sourceIP
```

Join Task

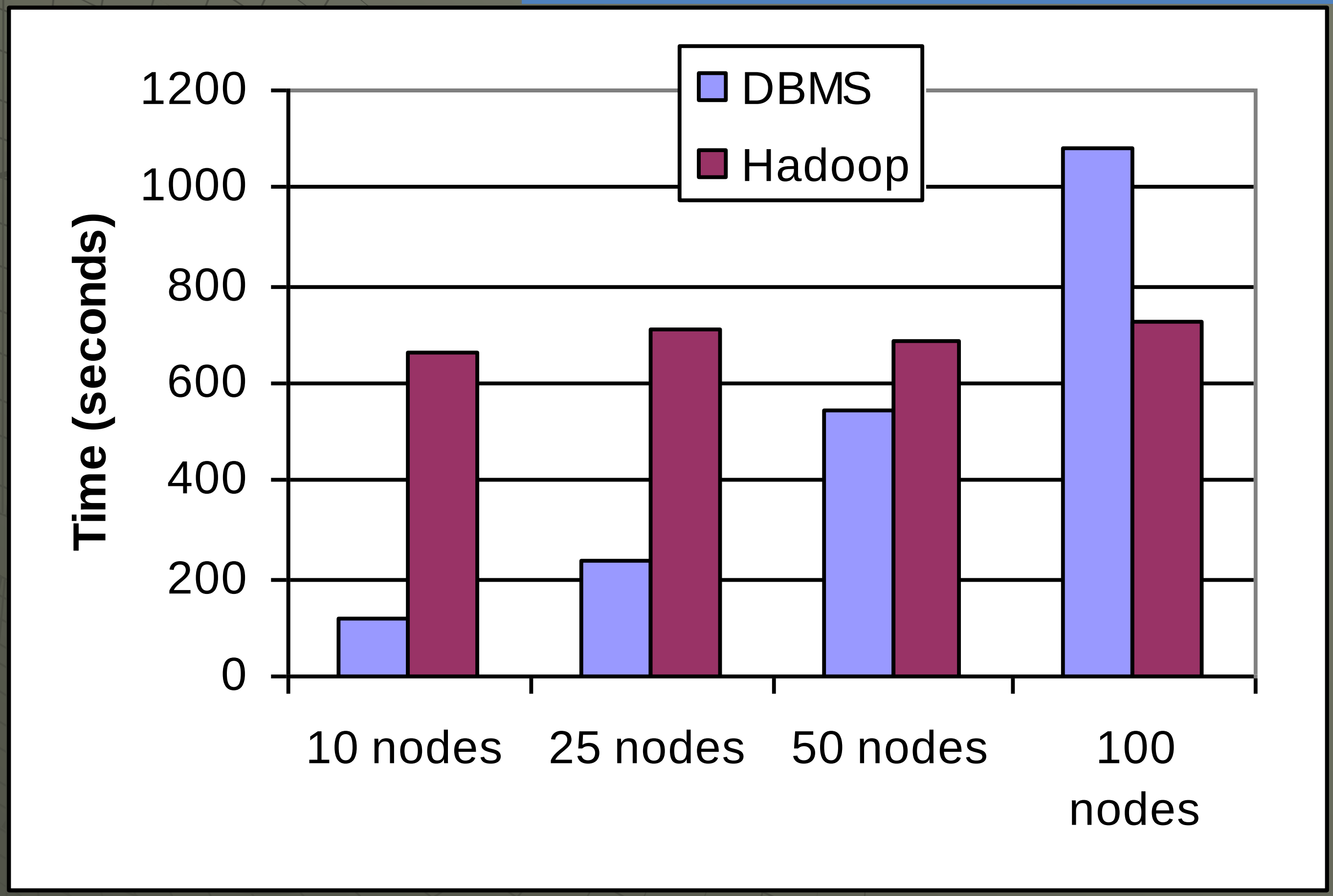
Database systems can co-partition by join key!



UDF Task

- Calculate PageRank over a set of HTML documents
- Performed via a UDF

DBMS clearly doesn't scale

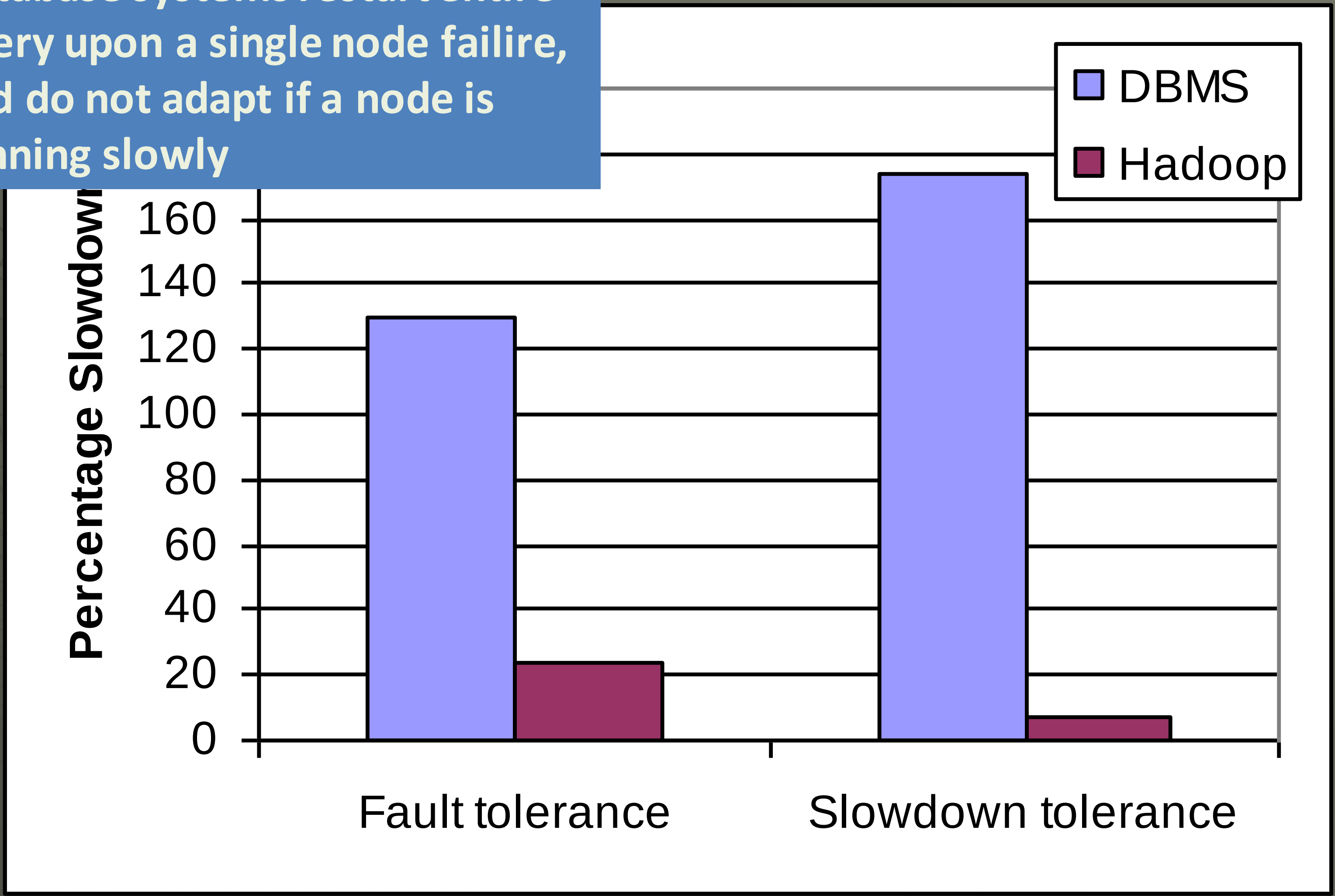


Scalabilty

- Except for DBMS-X load time and UDFs all systems scale near linearly
- BUT: only ran on 100 nodes
- As nodes approach 1000, other effects come into play
  - Faults go from being rare, to not so rare
  - It is nearly impossible to maintain homogeneity at scale

Fault Tolerance and Cluster Heterogeneity Results

Database systems restart entire query upon a single node failire, and do not adapt if a node is running slowly



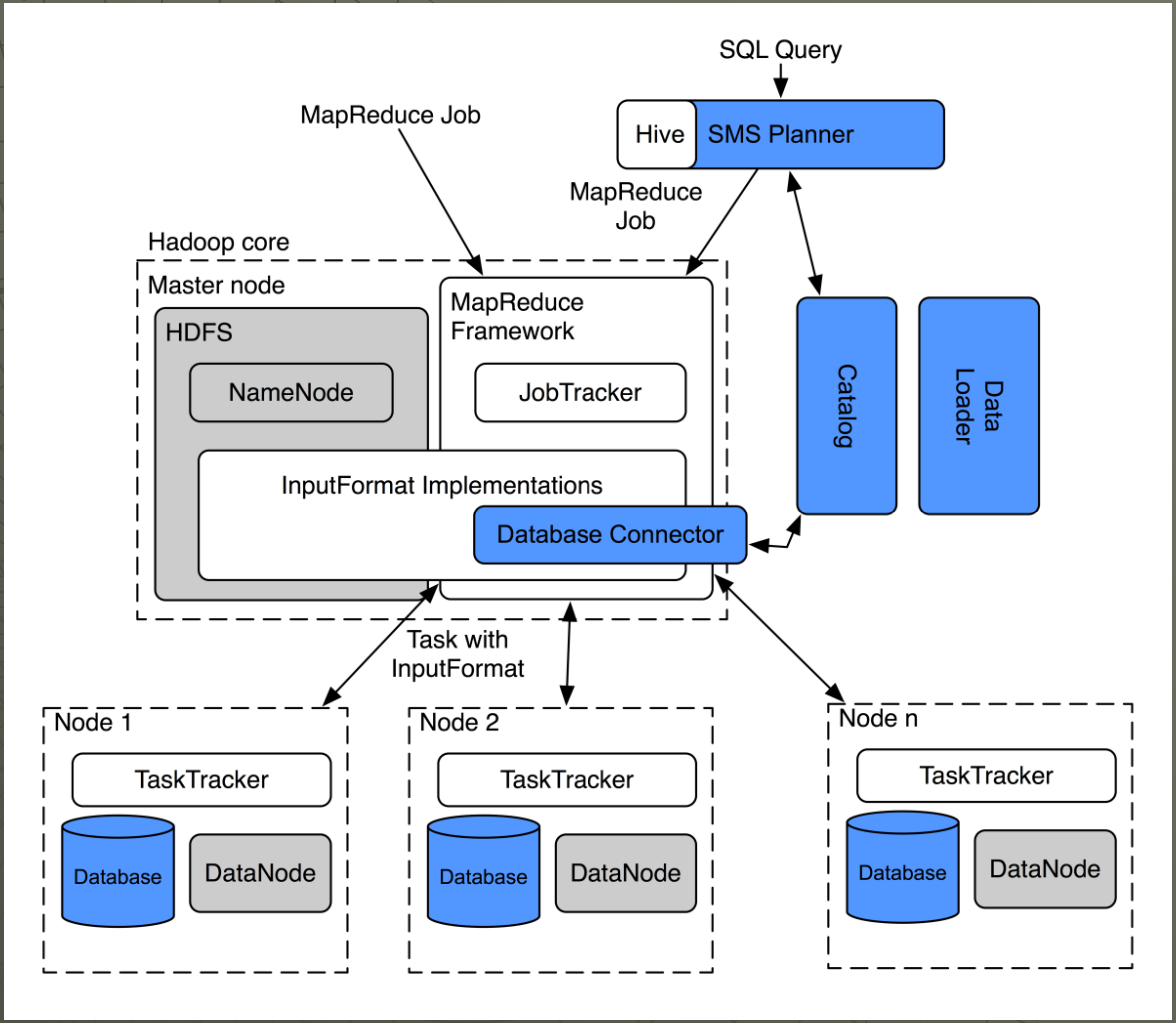
Benchmark Conclusions

- Hadoop is consistently more scalable
  - Checkpointing allows for better fault tolerance
  - Runtime scheduling allows for better tolerance of unexpectedly slow nodes
  - Better parallelization of UDFs
- Hadoop is consistently less efficient for structured, relational data
  - Reasons both fundamental and non-fundamental
  - Needs better support for compression and direct operation on compressed data
  - Needs better support for indexing
  - Needs better support for co-partitioning of datasets

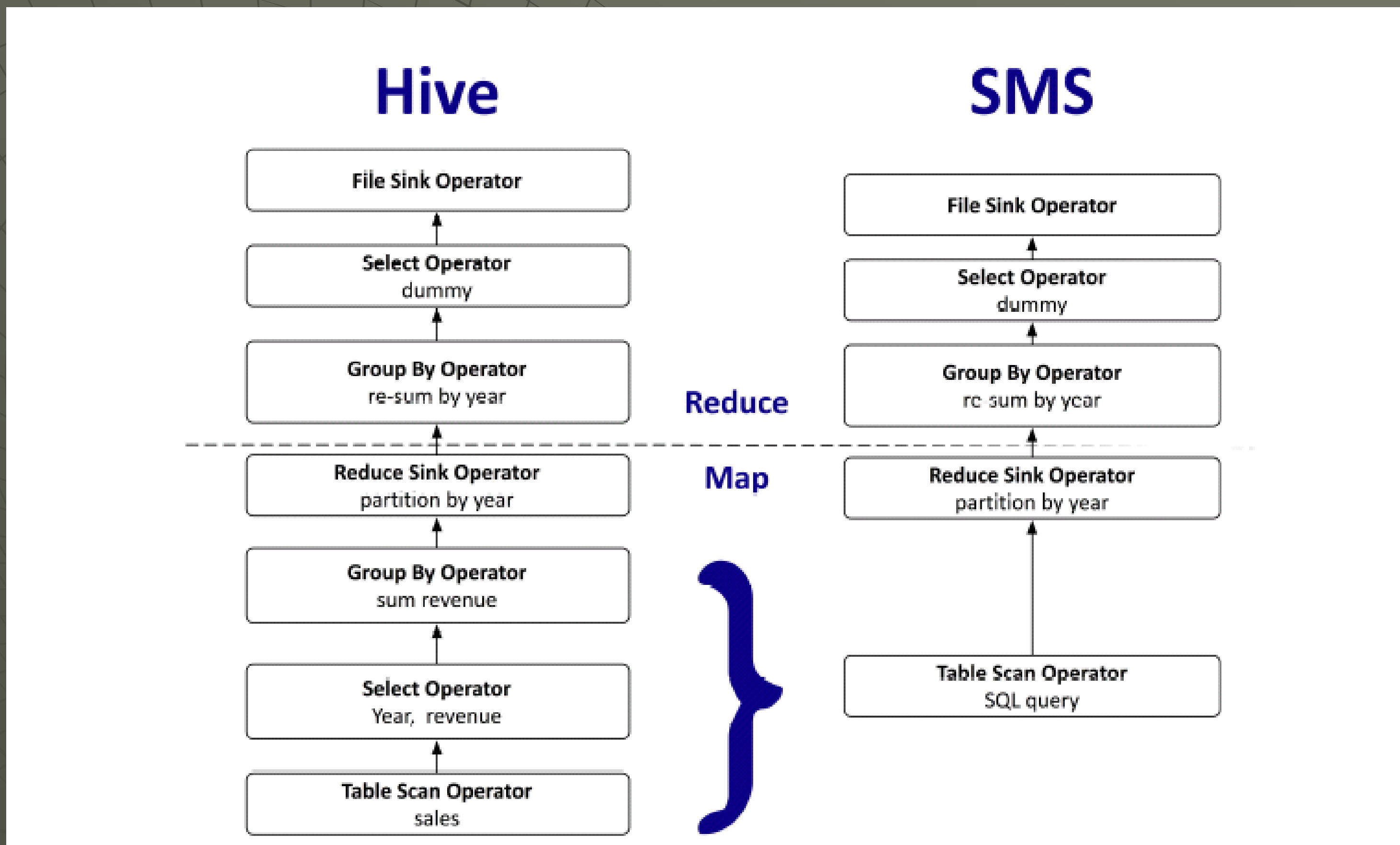
Best of Both Worlds Possible?

- Many of Hadoop's deficiencies not fundamental
  - Result of initial design for unstructured data
- HadoopDB: Use Hadoop to coordinate execution of multiple independent (typically single node, open source) database systems
  - Flexible query interface (accepts both SQL and MapReduce)
  - Open source (built using open source components)

HadoopDB Architecture



SMS Planner

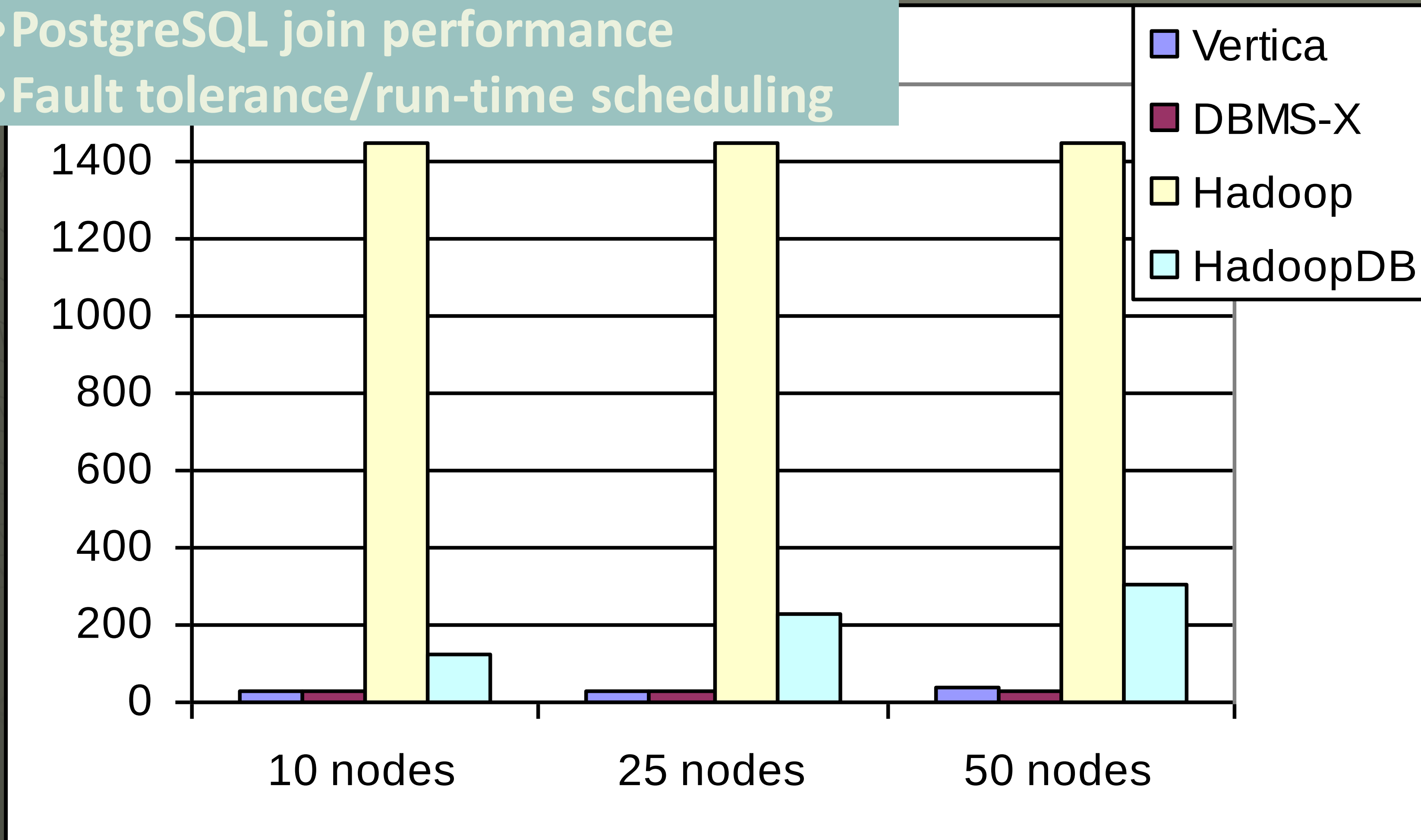


SELECT YEAR(saleDate), SUM(revenue) FROM sales GROUP BY YEAR(saleDate);

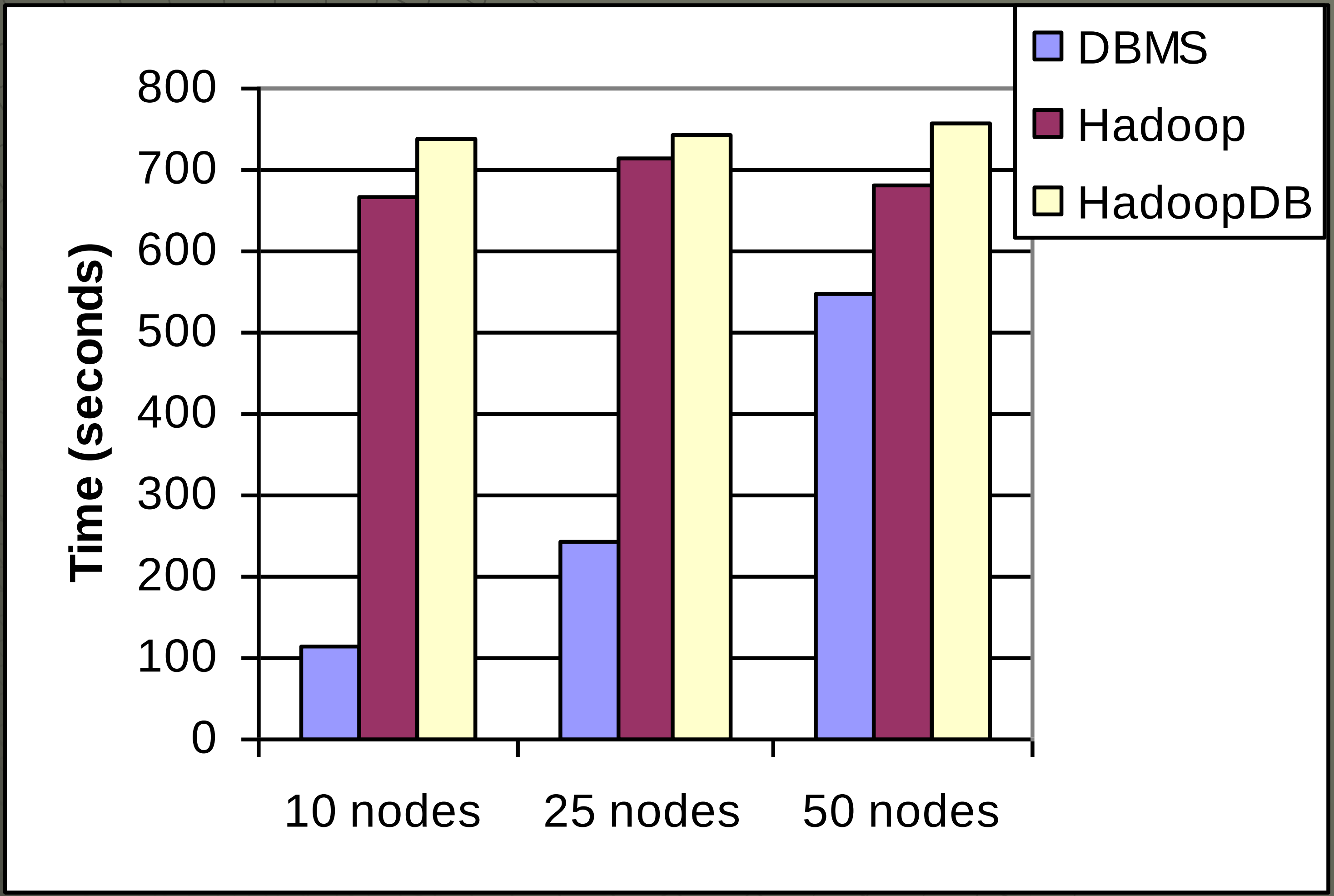
HadoopDB Experiments

- VLDB 2009 paper ran same Stonebraker Web analytics benchmark
- Used PostgreSQL as the DBMS storage layer

- HadoopDB must faster than Hadoop
- Doesn’t quite match the database systems in performance
- Hadoop start-up costs
- PostgreSQL join performance
- Fault tolerance/run-time scheduling

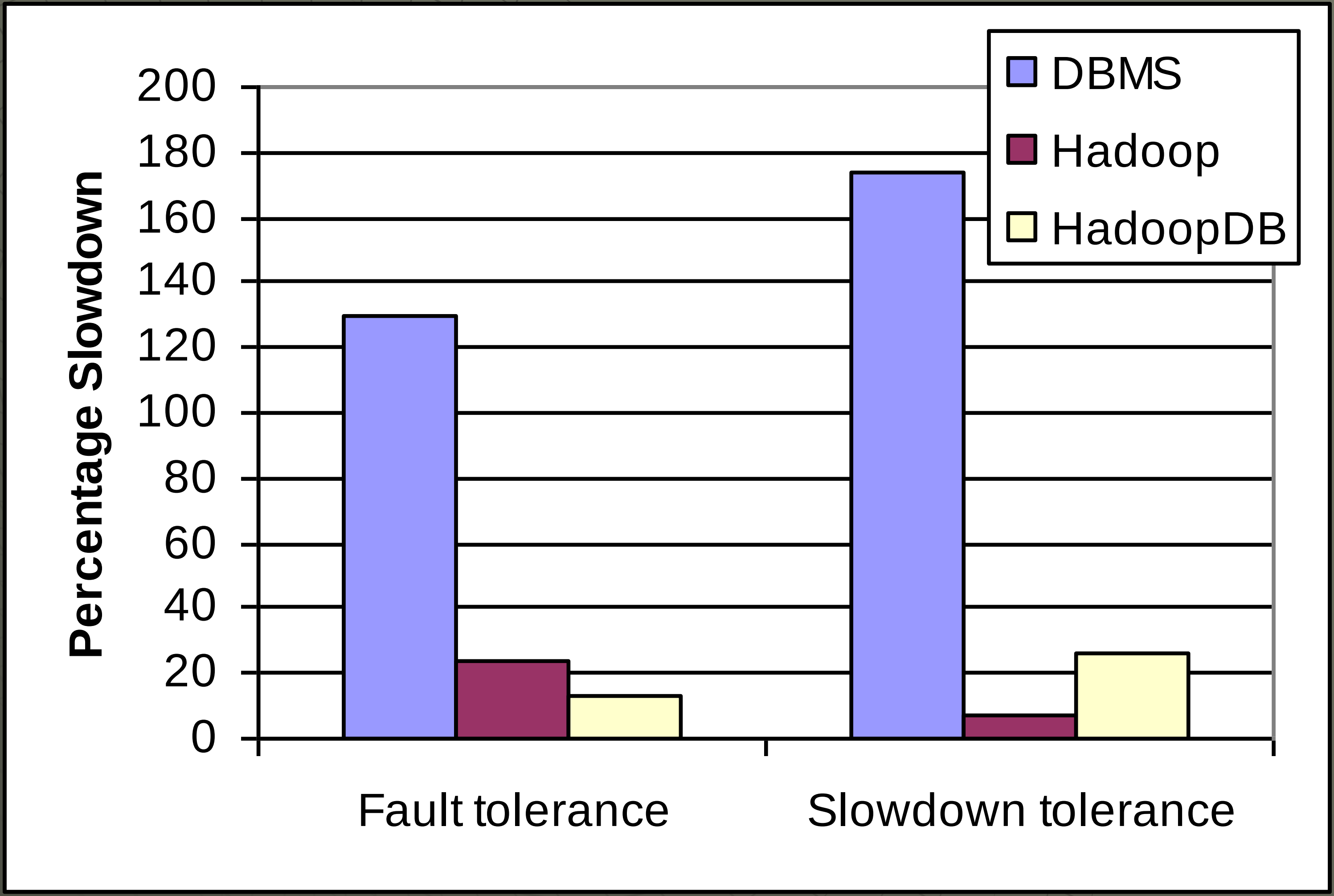


UDF Task



DBMS
Hadoop
HadoopDB
Time (seconds)
800
700
600
500
400
300
200
100
0
10 nodes
25 nodes
50 nodes

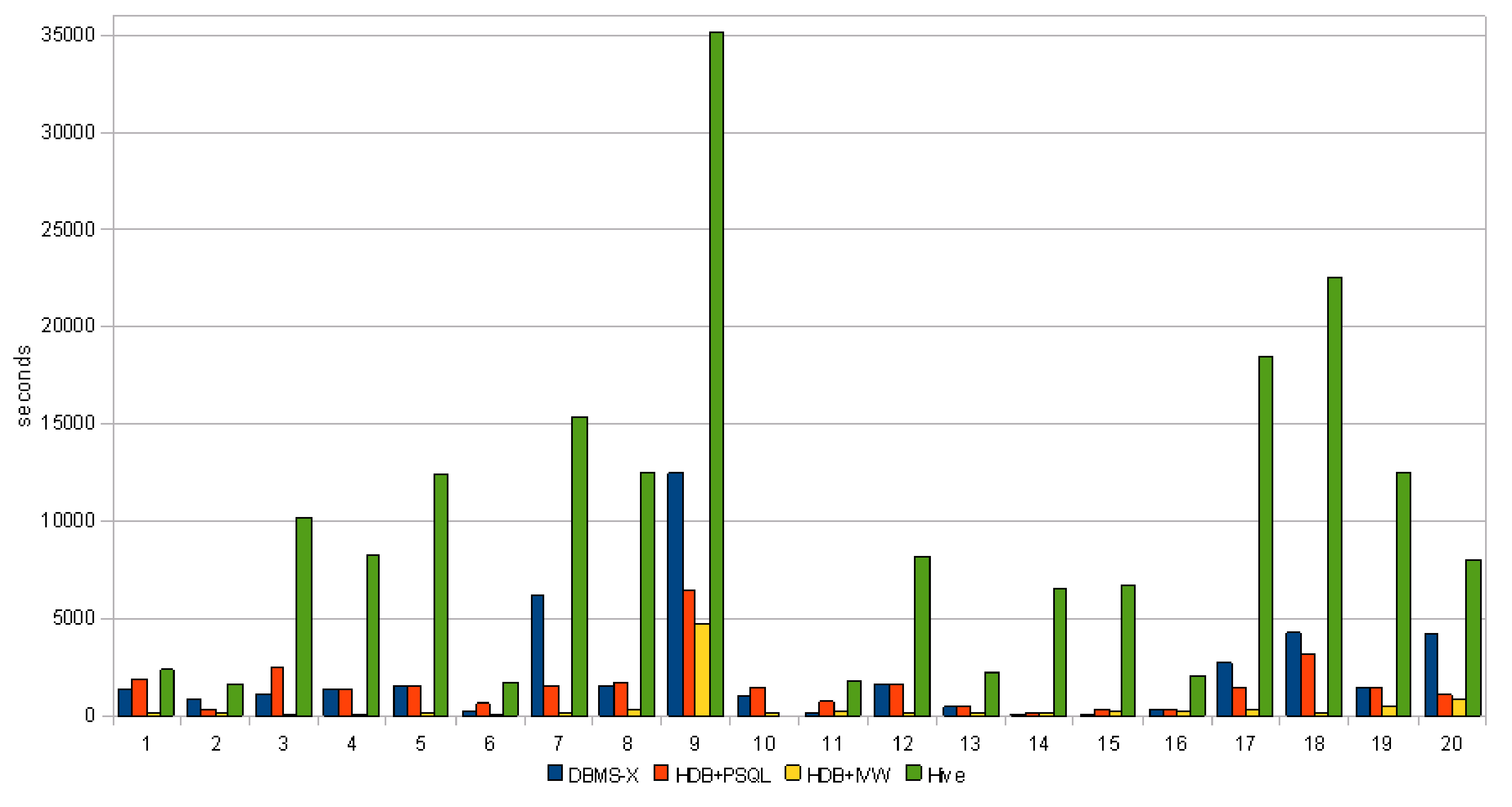
Fault Tolerance and Cluster Heterogeneity Results



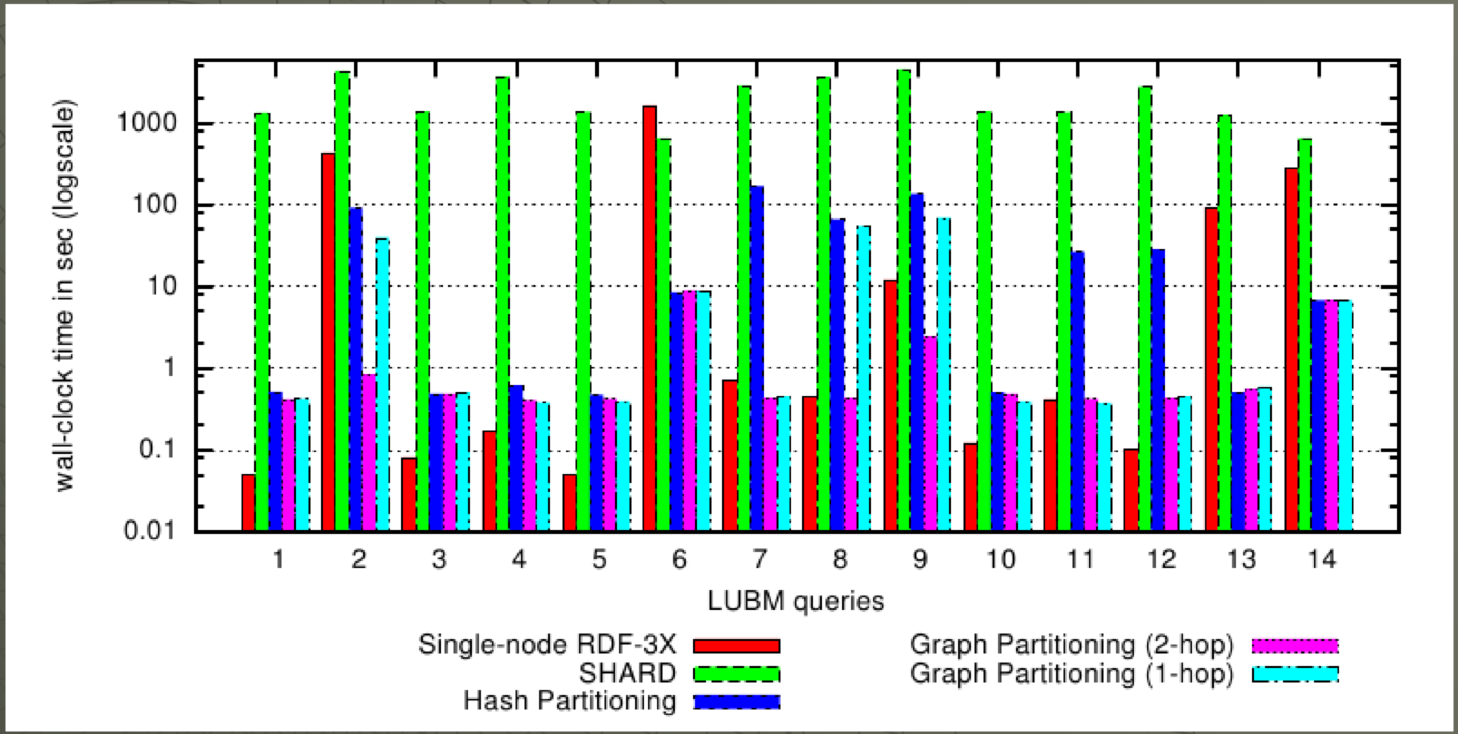
HadoopDB: Current Status

- Recently commercialized by Hadapt
  - Raised $9.5 million in venture capital
- SIGMOD 2011 paper benchmarking HadoopDB on TPC-H data
  - Added various other techniques
    - Column-store storage
    - 4 different join algorithms
    - Referential partitioning
- VLDB 2011 paper on using HadoopDB for graph data (with RDF-3X for storage)

TPC-H Benchmark Results



Graph Experiments



Invisible Loading

- Data starts in HDFS
- Data is immediately available for processing (immediate gratification paradigm)
- Each MapReduce job causes data movement from HDFS to database systems
- Data is incrementally loaded, sorted, and indexed
- Query performance improves “invisibly”

Conclusions

- Parallel database systems can be used for many data intensive tasks
  - Scalability can be an issue at extreme scale
  - Parallelization of UDFs can be an issue
- Hadoop is becoming increasingly popular and more robust
  - Free and open source
  - Great scalability and flexibility
  - Inefficient on structured data
- HadoopDB trying to get best of worlds
  - Storage layer of database systems with parallelization and job scheduling layer of Hadoop
- Hadapt is improving the code with all kinds of stuff that researchers don’t want to do
  - Full SQL support (via SMS planner)
  - Speed up (and automate) replication and loading
  - Easier deployment and managing
  - Automatic repartitioning about node addition/subtraction