Data Loading Challenges when Transforming Hadoop into a Parallel Database System

Daniel Abadi

Yale University

March 19th, 2013

Twitter: @daniel_abadi

Overview of Talk

- Motivation for HadoopDB
- Overview of HadoopDB
- Lessons learned from commercializing HadoopDB into Hadapt
- Ideas for overcoming the loading challenge (invisible loading)

- EDBT 2013 paper on invisible loading that is published alongside this talk can be found at: http://cs-www.cs.yale.edu/homes/dna/papers/invisibleloading.pdf

Big Data

- Data is growing (currently) faster than Moore's Law
  - Scale becoming a bigger concern
  - Cost becoming a bigger concern
  - Unstructured data "variety" becoming a bigger concern
- Increasing desire for incremental scale out on commodity hardware
  - Fault tolerance a bigger concern
  - Dealing with unpredictable performance a bigger concern
- Parallel database systems are the traditional solution
- A lot of excitement around Hadoop as a tool to help with these problems (initiated by Yahoo and Facebook)

Hadoop/MapReduce

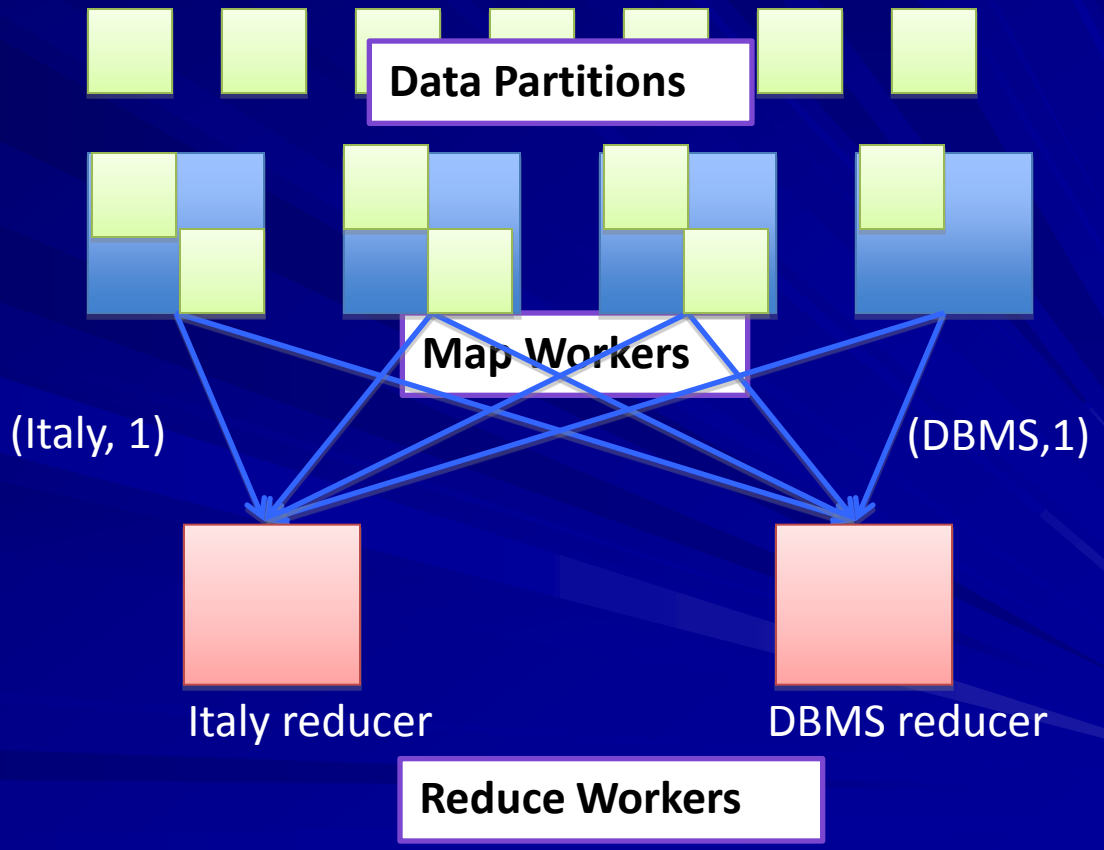
- Data is partitioned across N machines
  - Typically stored in a distributed file system (GFS/HDFS)
- On each machine n, apply a function, Map, to each data item d
  - Map(d) → {(key_1,$value_1$)} "map job"
  - Sort output of all map jobs on n by key
  - Send (key_1,$value_1$) pairs with same key value 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 “Italy” and “DBMS” in all documents

```
map(d):
    words = split(d,’ ‘)
    foreach w in words:
      if w == ‘Italy’
                  emit (‘Italy’, 1)
      if w == ‘DBMS’
                  emit (‘DBMS’, 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)

Similarities Between Hadoop and Parallel Database Systems

- 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

- Hadoop can operate on *in-situ* data, without requiring transformation or loading
- Schemas:
  - Hadoop doesn't require them, DBMSs do
  - Easy to write simple MR programs over raw data
- Indexes
  - Hadoop's built in support is primitive
- Declarative vs imperative programming
- Hadoop uses a run-time scheduler for fine-grained load balancing
- Hadoop checkpoints intermediate results for fault tolerance

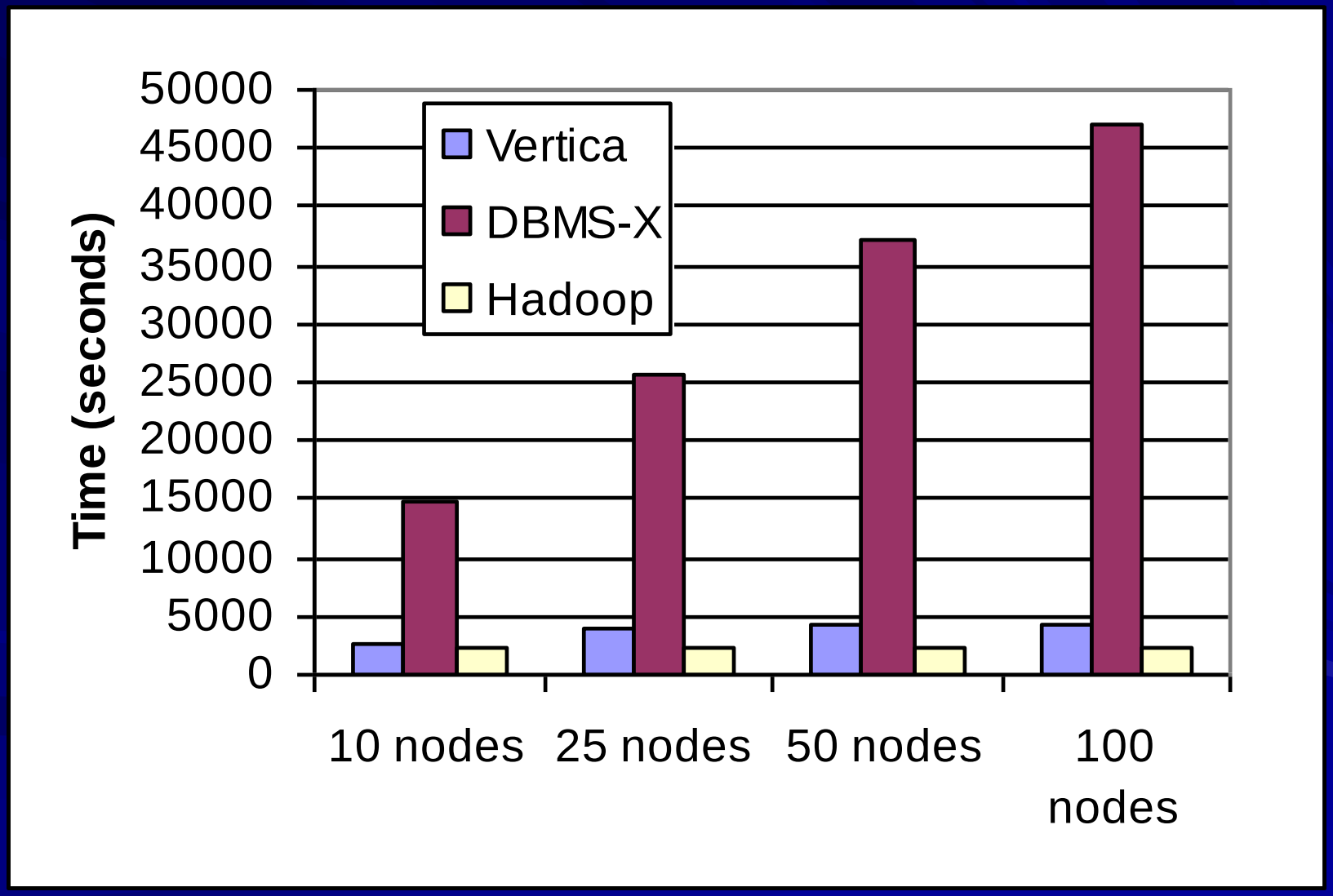
SIGMOD 2009 Paper

- Benchmarked Hadoop vs. 2 parallel database systems
  - Mostly focused on performance differences
  - Measured differences in load and query time for some common data processing tasks
  - Used Web analytics benchmark whose goal was to be representative of tasks that:
    - Both should excel at
    - Hadoop should excel at
    - Databases should excel at

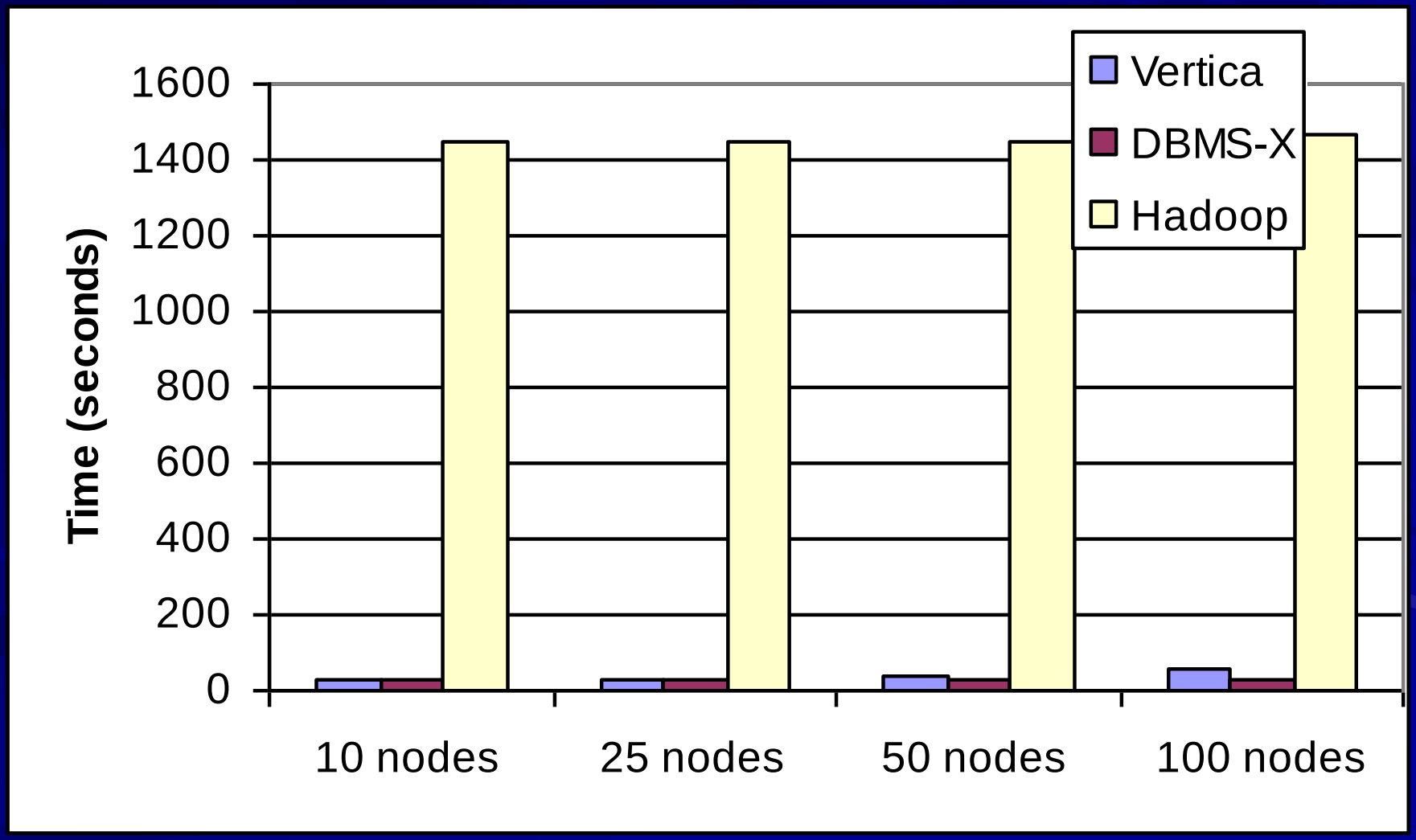
Hardware Setup

- 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

Loading


Time (seconds)
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
Vertica
DBMS-X
Hadoop
10 nodes
25 nodes
50 nodes
100 nodes

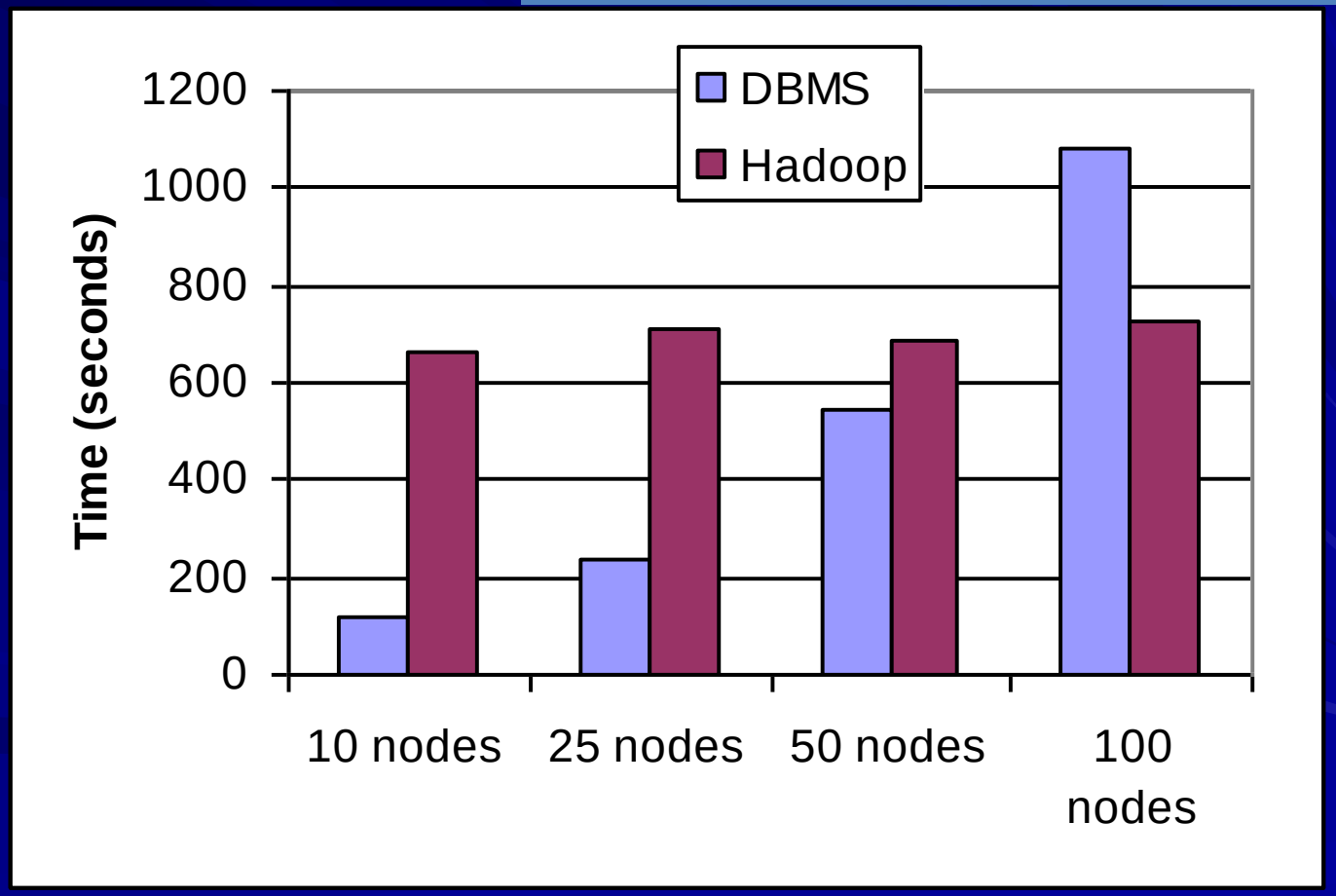
Join Task


Time (seconds)
1600
1400
1200
1000
800
600
400
200
0
10 nodes
25 nodes
50 nodes
100 nodes
Vertica
DBMS-X
Hadoop

UDF Task

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

DBMS clearly doesn't scale

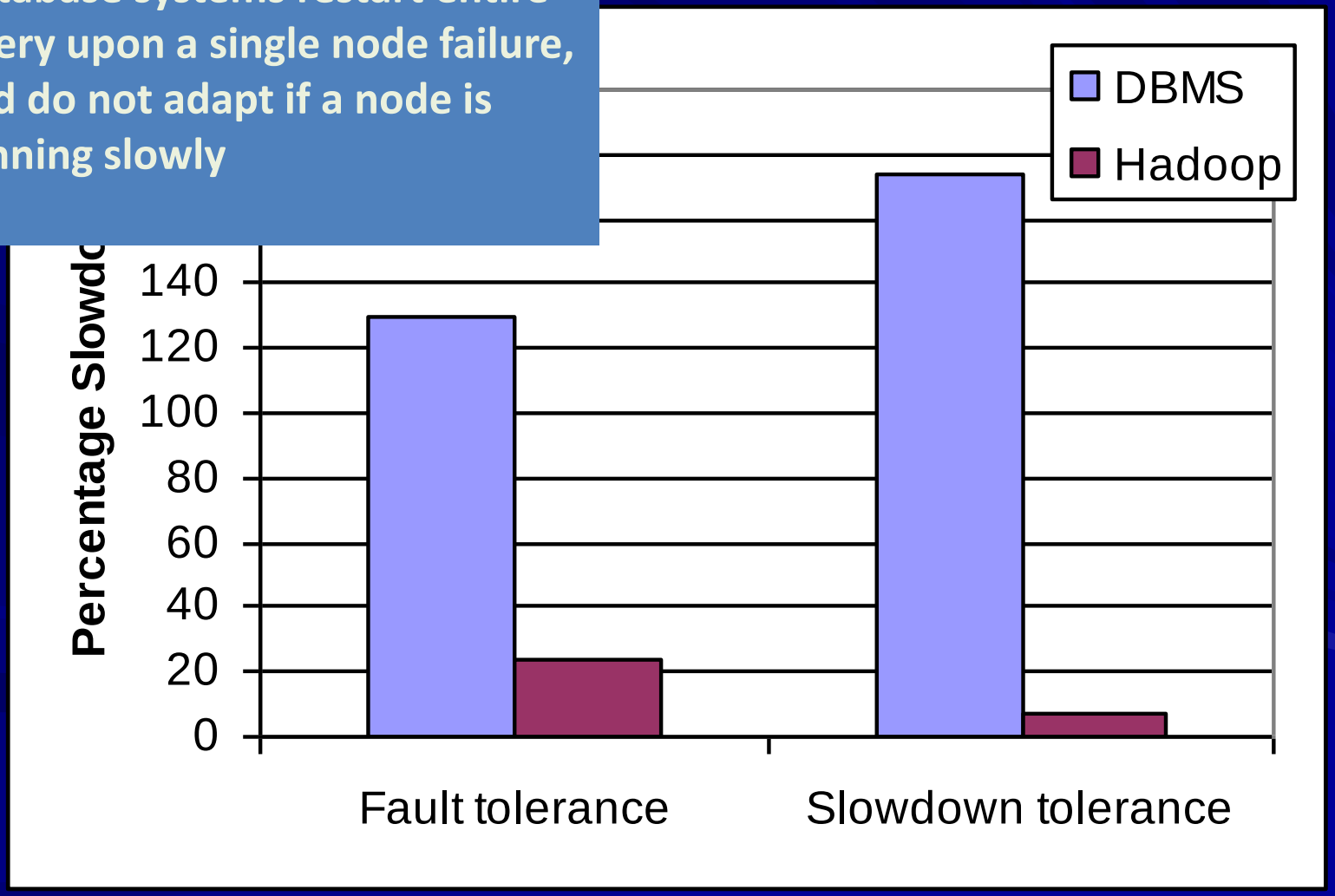


Scalability

- 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

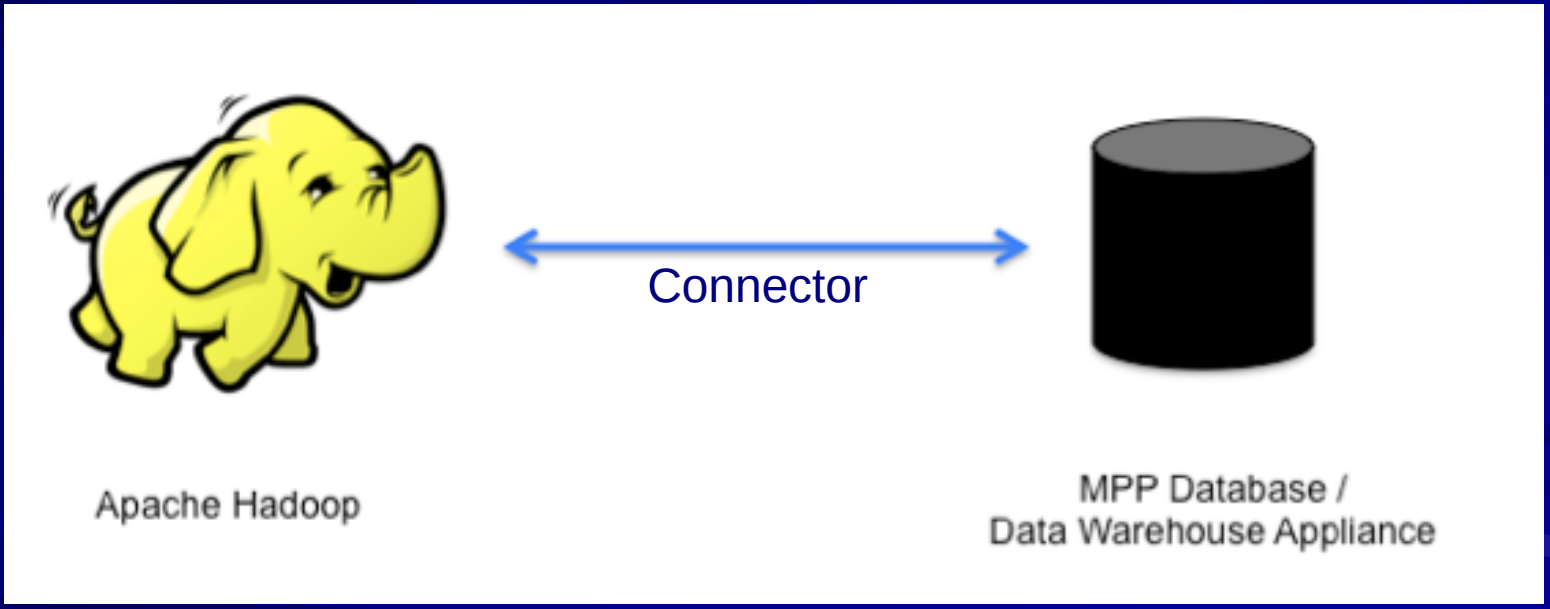
Database systems restart entire query upon a single node failure, and do not adapt if a node is running slowly



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 mostly 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?



Problems With the Connector Approach

- Network delays and bandwidth limitations
- Data silos
- Multiple vendors
- Fundamentally wasteful
  - Very similar architectures
    - Both partition data across a cluster
    - Both parallelize processing across the cluster
    - Both optimize for local data processing (to minimize network costs)

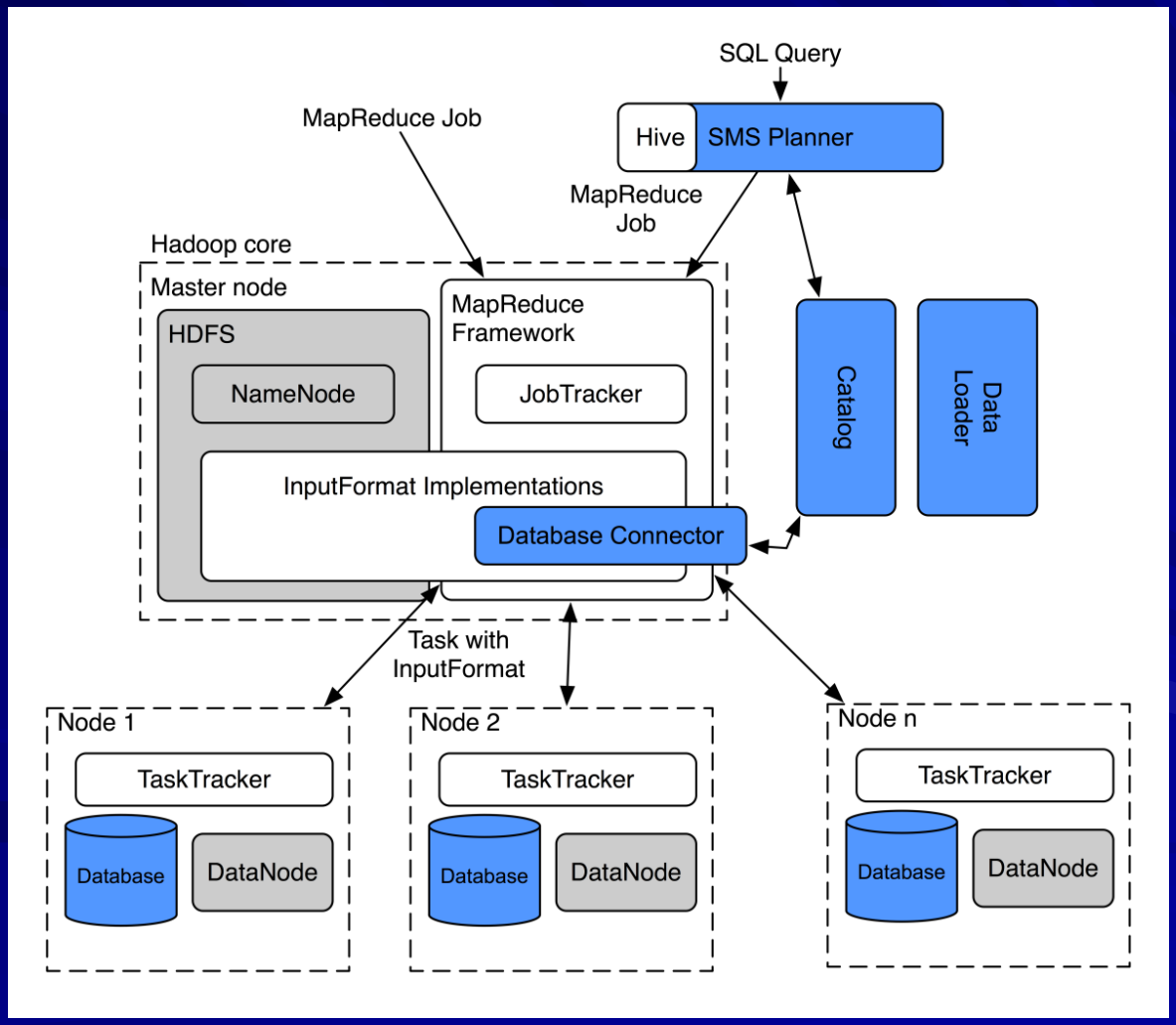
Unified System

- Two options:
  - Bring Hadoop technology to a parallel database system
    - Problem: Hadoop is more than just technology
  - Bring parallel database system technology to Hadoop
    - Far more likely to have impact

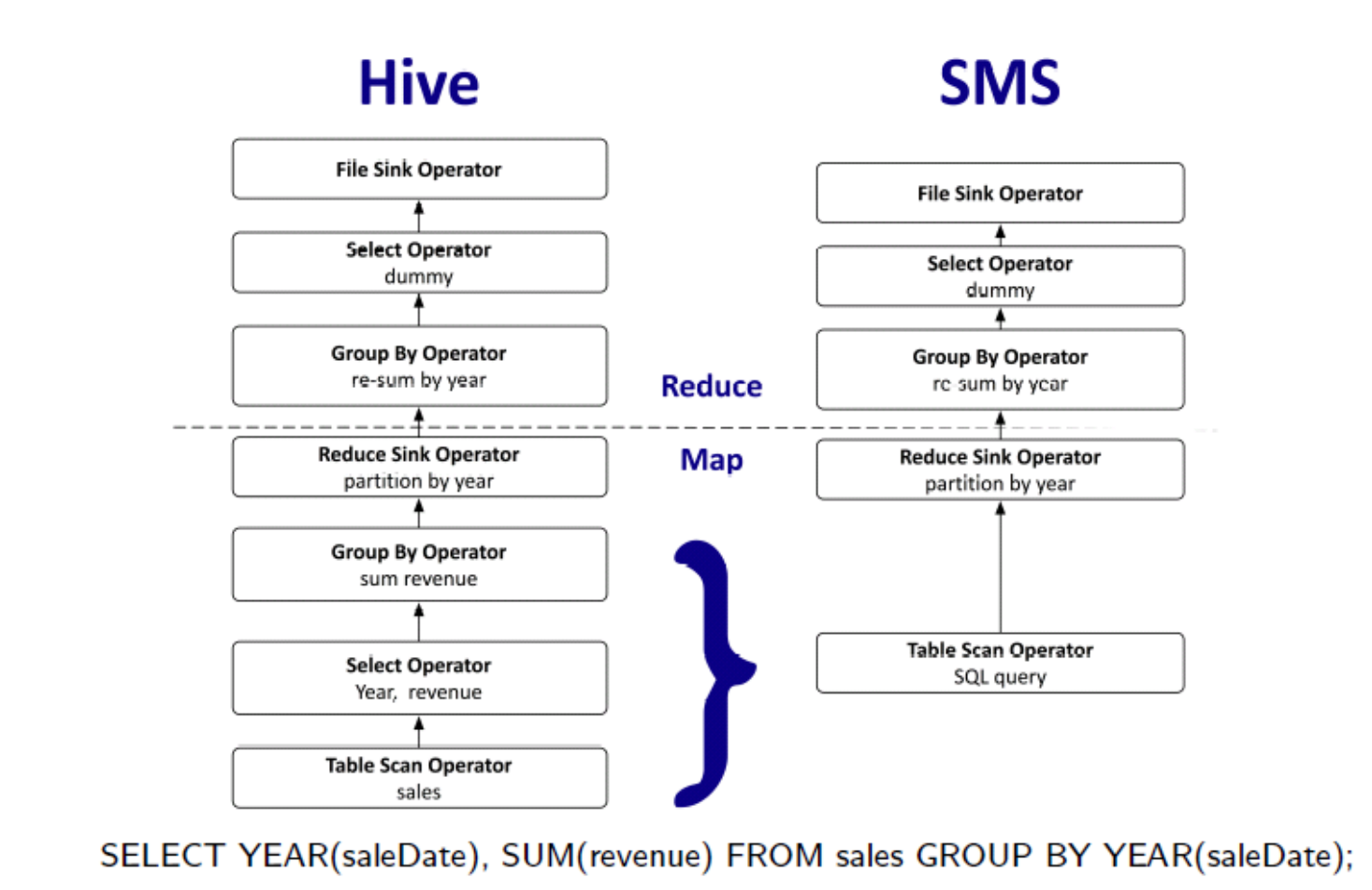
Adding DBMS Technology to Hadoop

- Option 1:
  - Keep Hadoop's storage and build parallel executor on top of it
    - Cloudera Impala (which is sort of a combination of Hadoop++ and NoDB research projects)
    - Need better Storage Formats (Trevni and Parquet are promising)
    - Updates and Deletes are hard (Impala doesn't support them)
  - Use relational storage on each node
    - Better support for traditional database management (including updates and deletes)
    - Accelerates "time to complete system"
    - We chose this option for HadoopDB

HadoopDB Architecture



SMS Planner

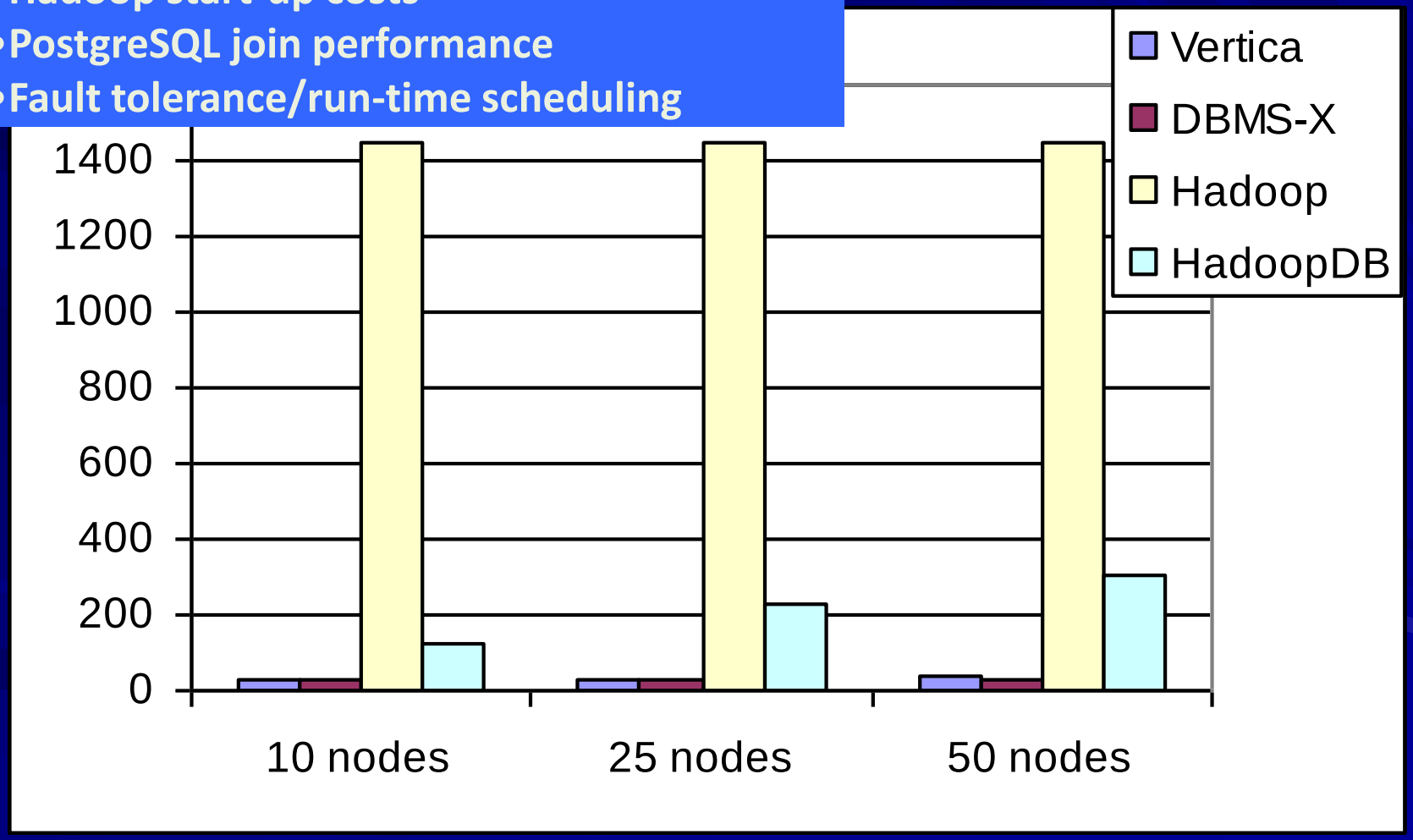

Hive
SMS
File Sink Operator
Select Operator
dummy
Group By Operator
re-sum by year
Reduce
Map
Reduce Sink Operator
partition by year
Group By Operator
sum revenue
Select Operator
Year, revenue
Table Scan Operator
sales
File Sink Operator
Select Operator
dummy
Group By Operator
re-sum by year
Reduce Sink Operator
partition by year
Table Scan Operator
SQL query
SELECT YEAR(saleDate), SUM(revenue) FROM sales GROUP BY YEAR(saleDate);

HadoopDB Experiments

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

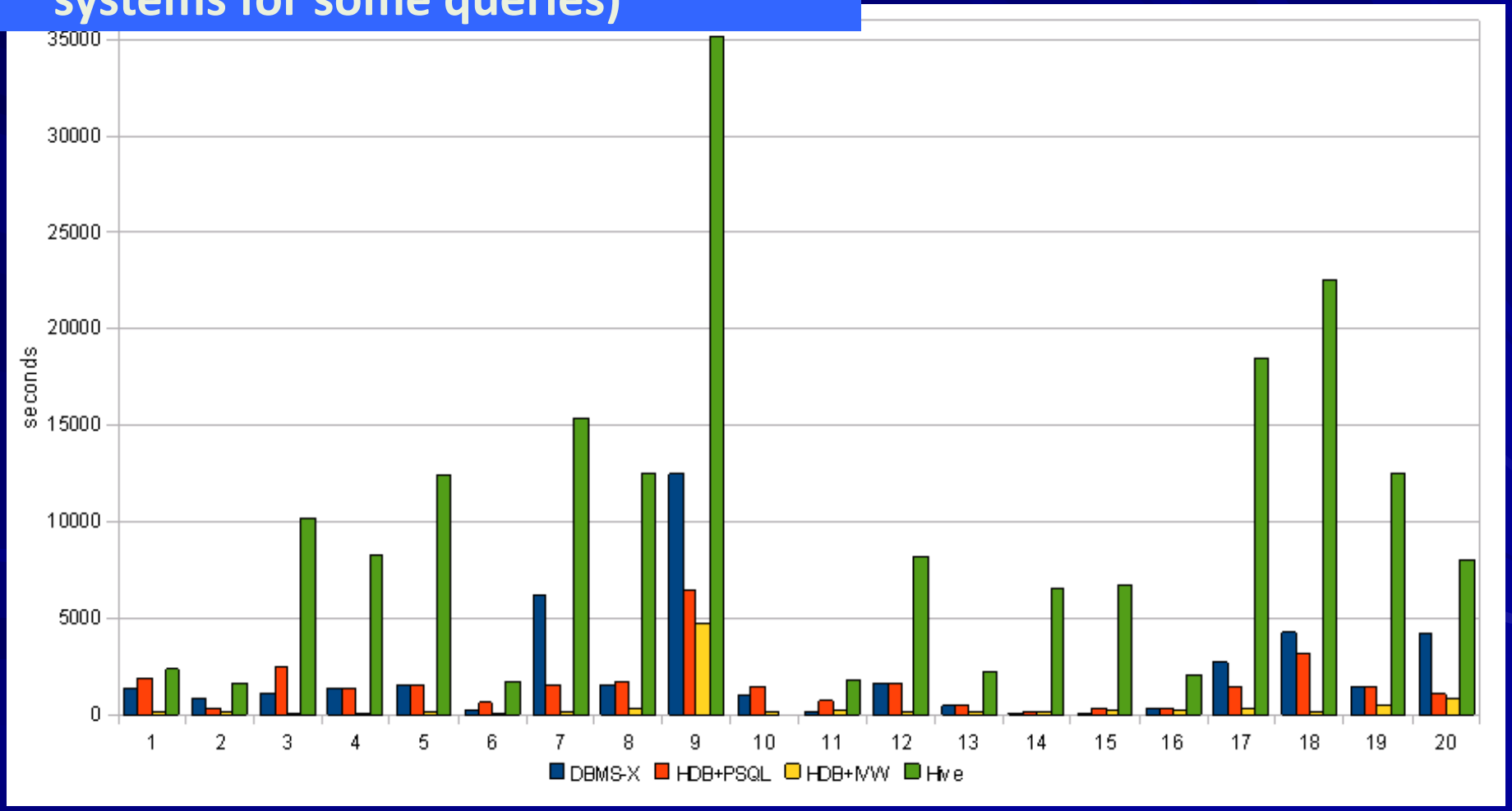
Join Task

- HadoopDB much faster than Hadoop
- In 2009, didn't quite match the database systems in performance (but see next slide)
- Hadoop start-up costs
- PostgreSQL join performance
- Fault tolerance/run-time scheduling

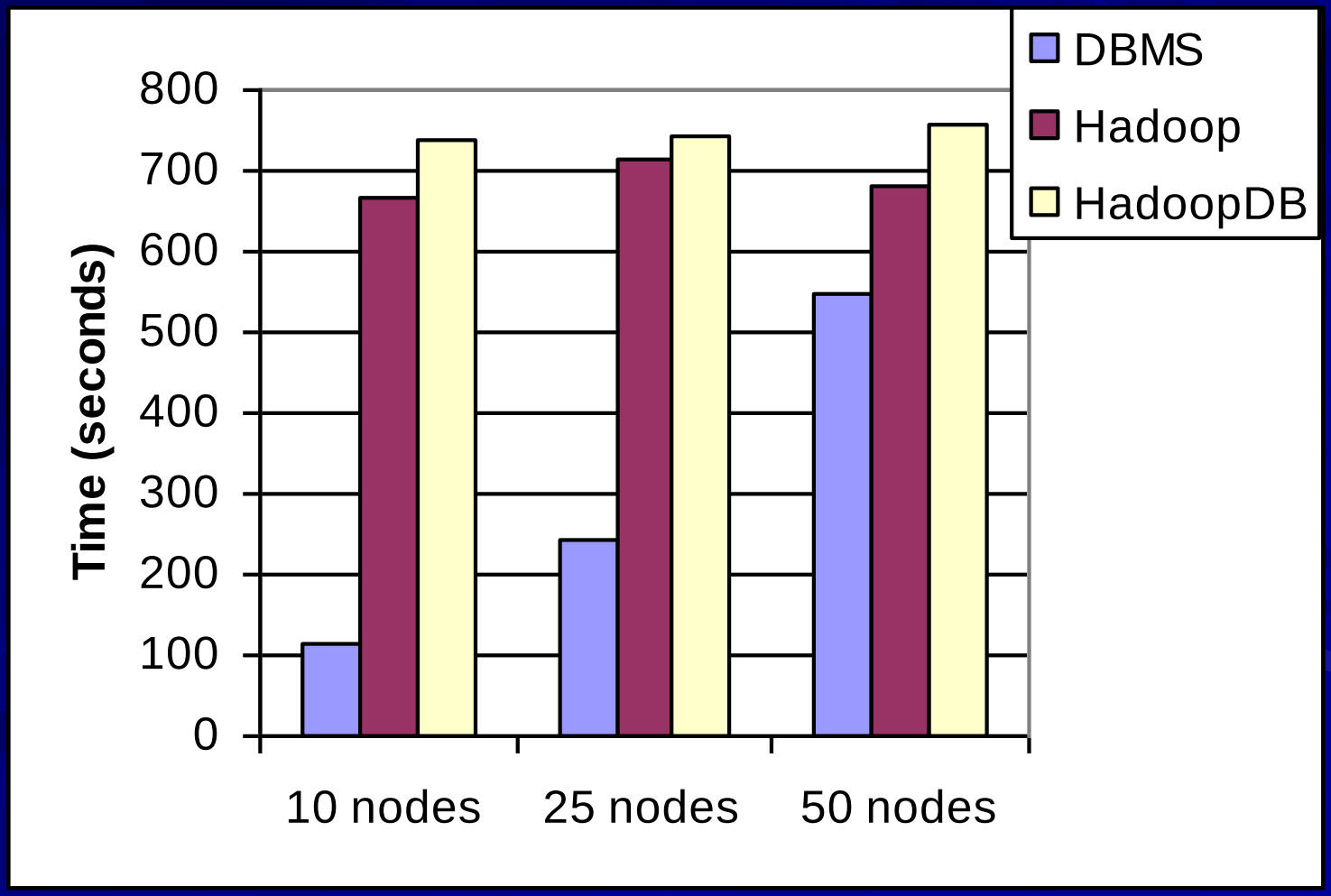


TPC-H Benchmark Results

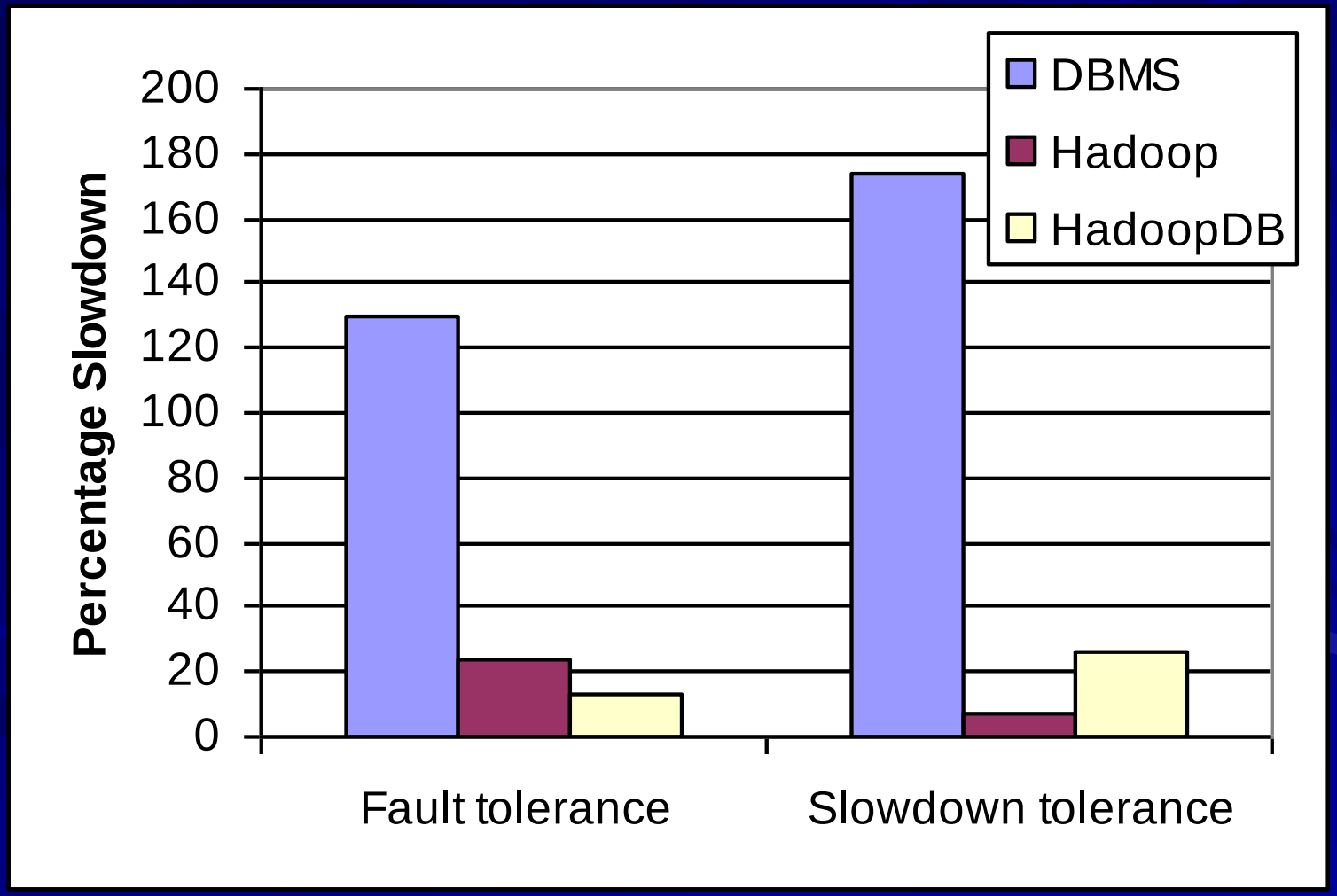
- More advanced storage and better join algorithms can lead to better performance (even beating database systems for some queries)



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



Hadapt

- Goal: Turn HadoopDB into enterprise-grade software so it can be used for real analytical applications
  - Launched with $1.5 million in seed money in 2010
  - Raised an additional $8 million in 2011
  - Raised an additional $6.75 million in 2012

Speaker has a significant financial interest in Hadapt

Hadapt: Lessons Learned

- Location matters
  - Impossible to scale quickly in New Haven, Connecticut
  - Company was forced to move to Cambridge (near other successful DB companies)
    - Personally painful for me

Hadapt: Lessons Learned

- Business side matters
  - Need intelligence and competence across the whole company (not just in engineering)
    - Decisions have to made all the time that have enormous effects on the company
  - Technical founders often find themselves doing non-technical tasks more than half the time

Hadapt: Lessons Learned

- Good enterprise software: more time on minutia than innovative new features
  - SQL coverage
  - Failover for high availability
  - Authorization / authentication
  - Disaster Recovery
  - Installer, upgrader, downgrader (!)
  - Documentation

Hadapt: Lessons Learned

- Customer input to the design process critical
  - Hadapt had no plans to integrate text search into the product until request from a customer
  - Today ~75% of Hadapt customers use the text search feature

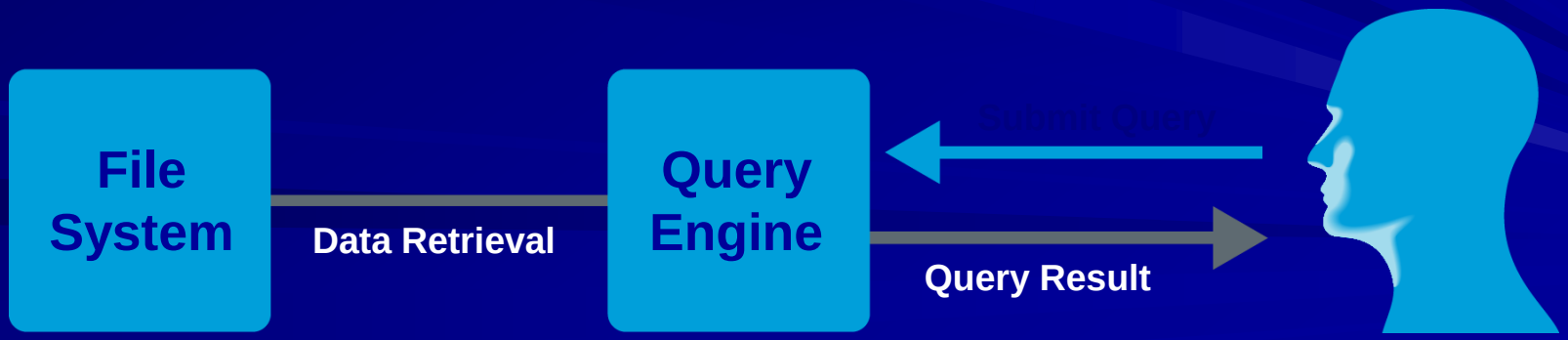
Hadapt: Lessons Learned

- Installation and load processes are critical
  - They are the customer's first two interactions with your product
  - Lead to the invisible loading idea
    - Discussed in the remaining slides of this presentation

If only we could make loading invisible …

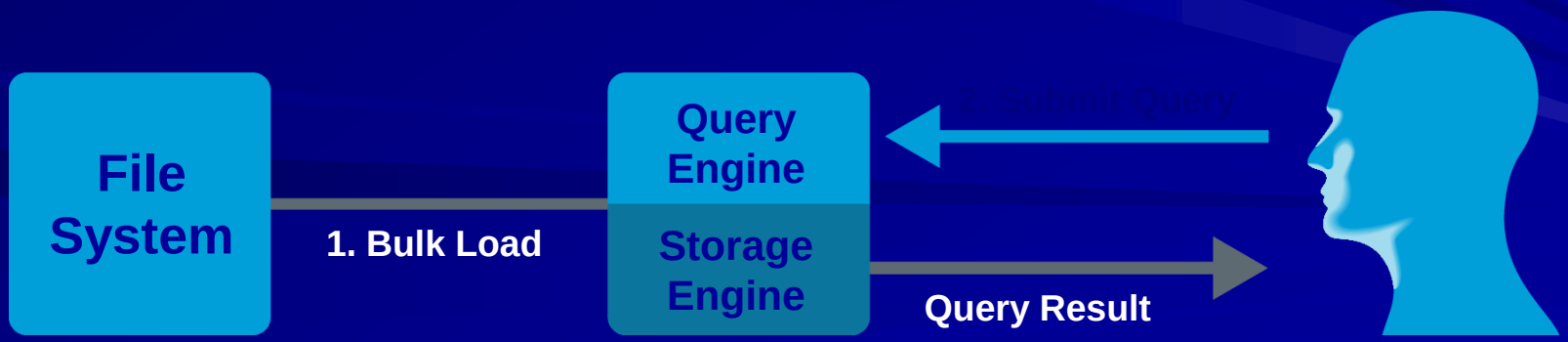
Review: Analytic Workflows

- Assume data already in file system (HDFS)
- Two workflows
  - Workflow #1: In-situ querying
    - e.g. Map Reduce, HIVE / Pig, Impala, etc
    - Latency optimization (time to first query answer)



Review: Analytic Workflows

- Assume data already in file system (HDFS)
- Two workflows
  - Workflow #2: Querying after bulk load
    - E.g. RDBMS, Hadapt
    - Storage and retrieval optimization for throughput (repeated queries)
    - Other benefits: data validation and quality control

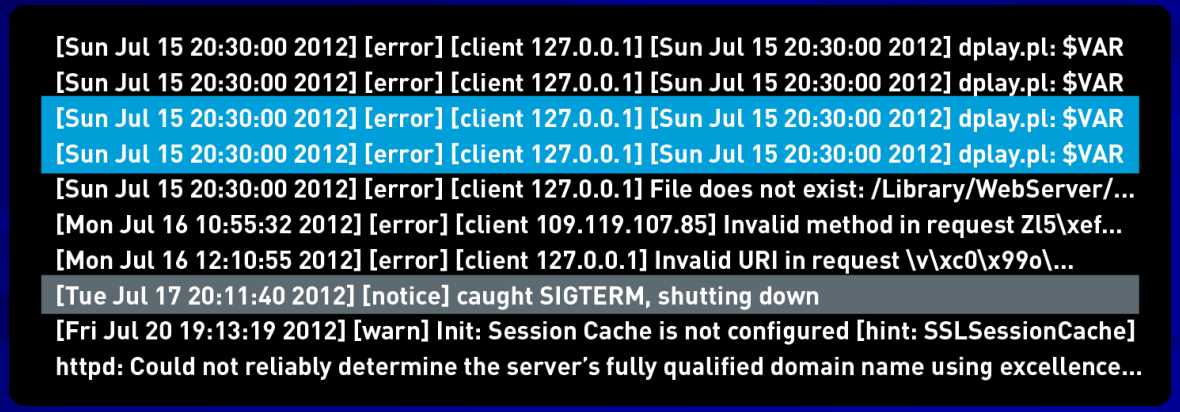


Immediate Versus Deferred Gratification

VS.

Use Case Example

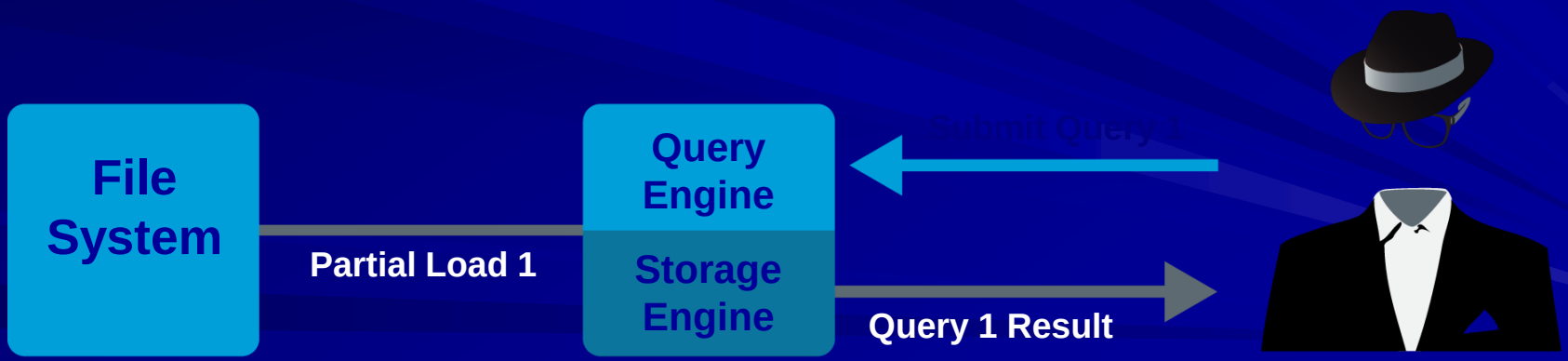
- Context-Dependent Schema Awareness
- Different analysts know the schema of what they are looking for and don't care about other log messages



Message field varies depending on application

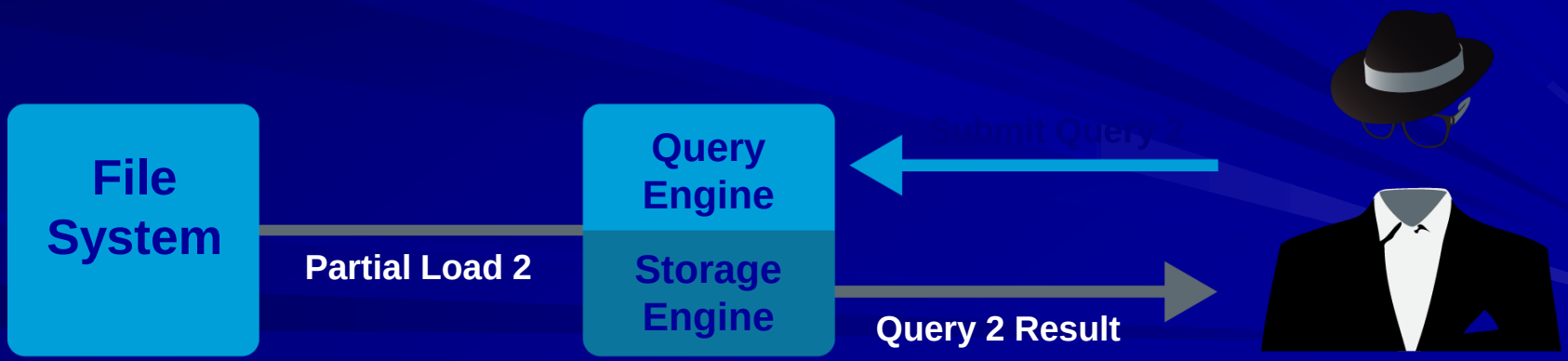
Workload-Based Data Loading

- Invisible loading makes workload-directed decisions on:
  - When to bulk load which data
  - By piggy-backing on queries



Workload-Based Data Loading

- Invisible loading makes workload-directed decisions on:
  - When to bulk load which data
  - By piggy-backing on queries



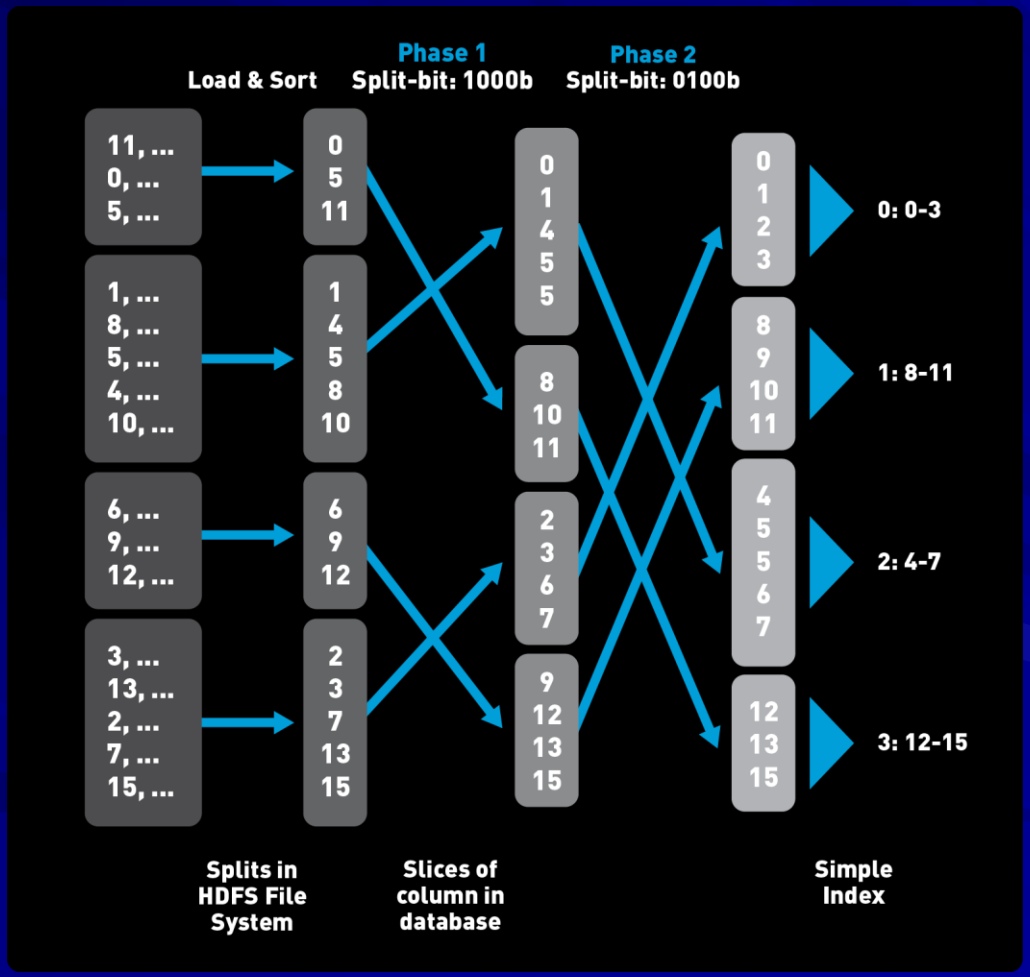
Workload-Based Data Loading

- Invisible loading makes workload-directed decisions on:
  - When to bulk load which data
  - By piggy-backing on queries
- Benefits
  - Eliminates manual workflows on data modeling and bulk load scripting
  - Optimizes query performance over time
  - Boosts productivity and performance

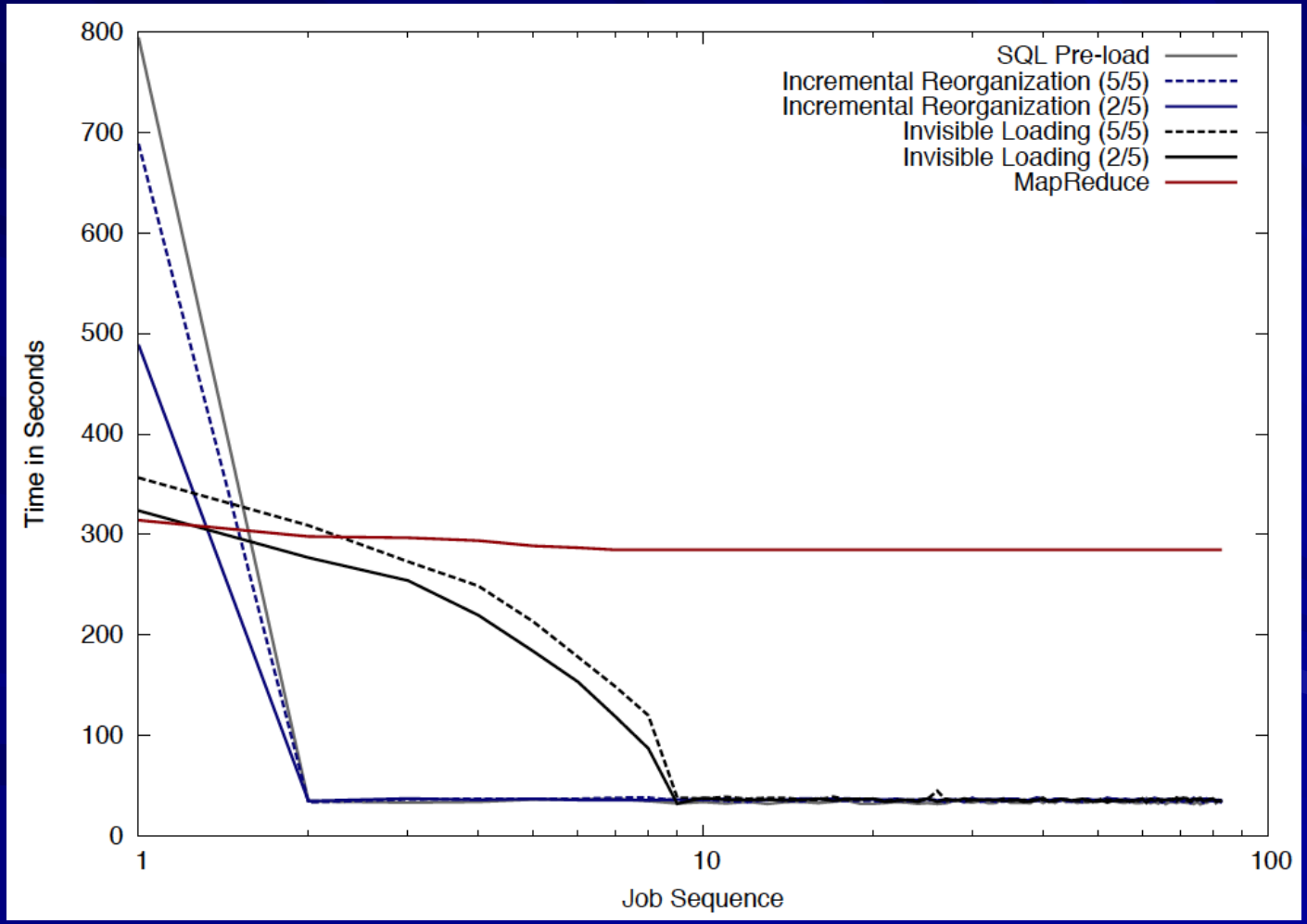
Interface and User Experience Design

- Structured parser interface for mapper
  - *getAttribute(int index)*
  - Similar to PigLatin
- Minimize load's impact on query performance
  - Load a subset of rows (based on HDFS file splits)
  - Load size tunable based on query performance impact
- Optional: load a subset of columns
  - Useful for columnar file formats

Physical Tuning: Incremental Reorganization



Performance Study



Conclusion

- Invisible loading: Immediate gratification + long term performance benefits
- Applicability beyond HDFS
- Future work
  - Automatic data schema / format recognition
  - Schema evolution support
  - Decouple load from query
    - Workload-sensitive loading strategies