HRDBMS:
A NewSQL Database for Analytics

Jason Arnold

Advisors: Ioan Raicu, Boris Glavic

Accepted as a short paper + poster for IEEE Cluster 2015

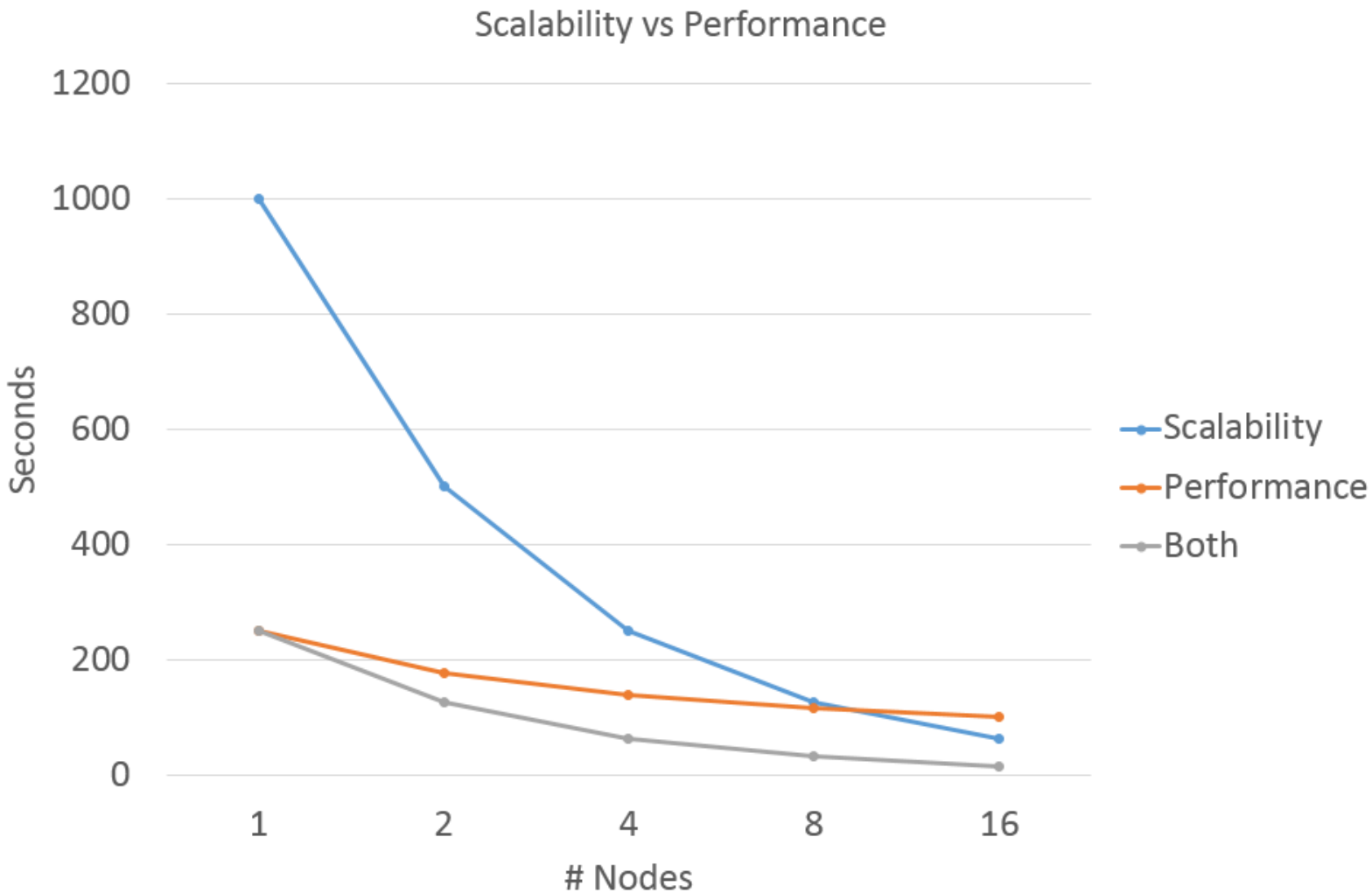
Agenda

- Background and motivation
- HRDBMS design goals
- HRDBMS architecture
- HRDBMS overview
- Performance results
- Conclusions
- Future work

Background and Motivation

- It is not uncommon to see 100TB+ data warehouses
  - … and data continues to grow
- However, the most scalable traditional MPP relational databases only scale well to a few hundred nodes
- At that cluster size and data volume, we can see queries that take hours, with no existing technology capable of speeding them up further
  - … and this only continues to get worse
- Modern databases like Hive (Hadoop) and Spark SQL (Spark) can scale to larger clusters, but per-node performance is very poor
  - It would take a very large Hive or Spark cluster to help

Background and Motivation



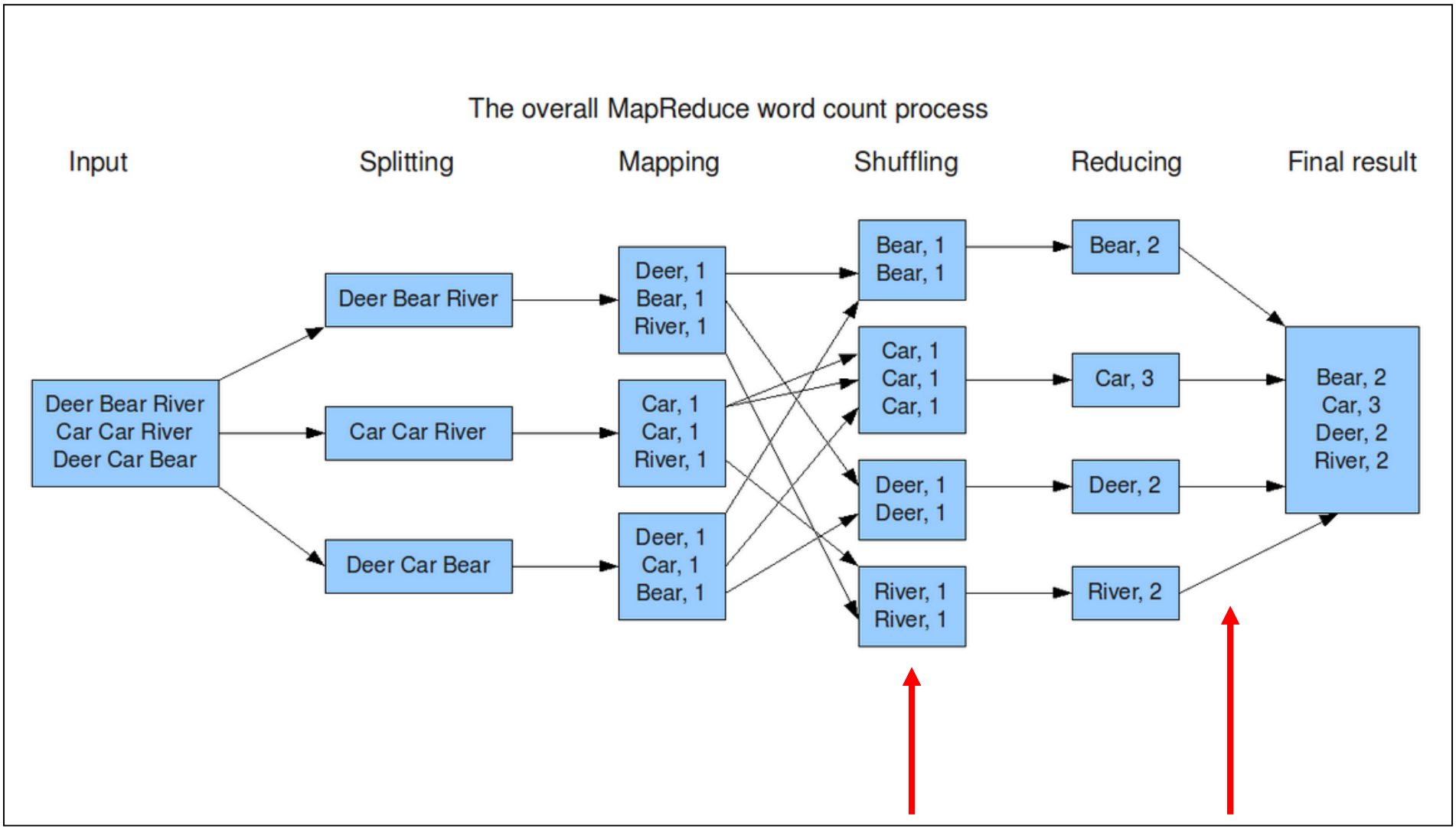
Diagnosing the Problem

- If we look into why per-node performance is so poor for databases like Hive or Spark, we find…
- The performance problem is due to the model and restrictions of the underlying framework
  - Queries are broken up into multiple jobs
  - Data is passed between jobs by externalizing to disk (HDFS)
  - Unnecessary sorts and externalization take place within jobs
  - Blocking operations prevent data from being pipelined
  - In general, being reliant on the general-purpose API of the framework/platform

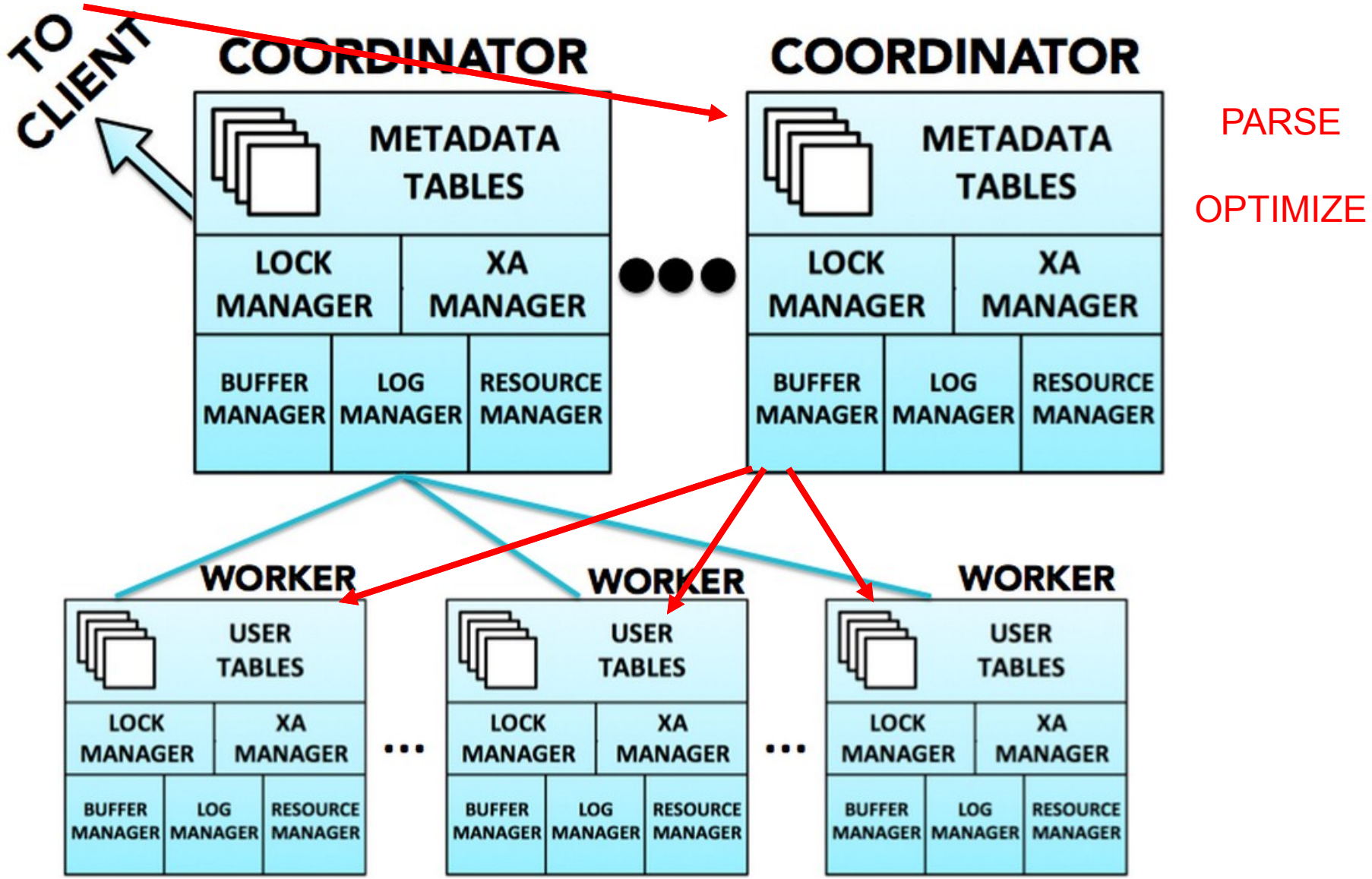
Solving the Problem

- This presents us with a possible solution…
  - If we create a custom framework that is modeled after MapReduce
  - But, removes the identified bottlenecks
  - And, is specifically designed for SQL analytics workloads
  - Can we improve on the per-node performance?
  - While maintaining the scalability of the MapReduce model?

MapReduce Overview



HRDBMS Architecture



How Does HRDBMS Accomplish These Goals?

- Primary goals
  - We made the observation that most of the reduce operations required can be rewritten as map operations
  - So rather than forcing map operations to be followed by reduce operations, we allow multiple map operations to be chained together via shuffle operations
  - This allows us to convert a single SQL query into a single HRDBMS job
  - No materialization to disk to pass data between jobs
  - Allows us to pipeline the entire job

How Does HRDBMS Accomplish These Goals?

- Primary goals
  - The shuffle operation in Hadoop and Spark is a big bottleneck
    - Hadoop – Blocking operation, always sorts and writes to disk
    - Spark – Blocking operation, sometimes sorts, always writes to disk
  - HRDBMS – non-blocking shuffle, never sorts, never writes to disk
    - Makes pipelining even more successful
  - HRDBMS also implements a novel hierarchical shuffle
    - When the number of neighbor nodes to communicate with becomes large, HRDBMS uses a hierarchical model of network communications to make the shuffle more efficient

How Does HRDBMS Accomplish These Goals?

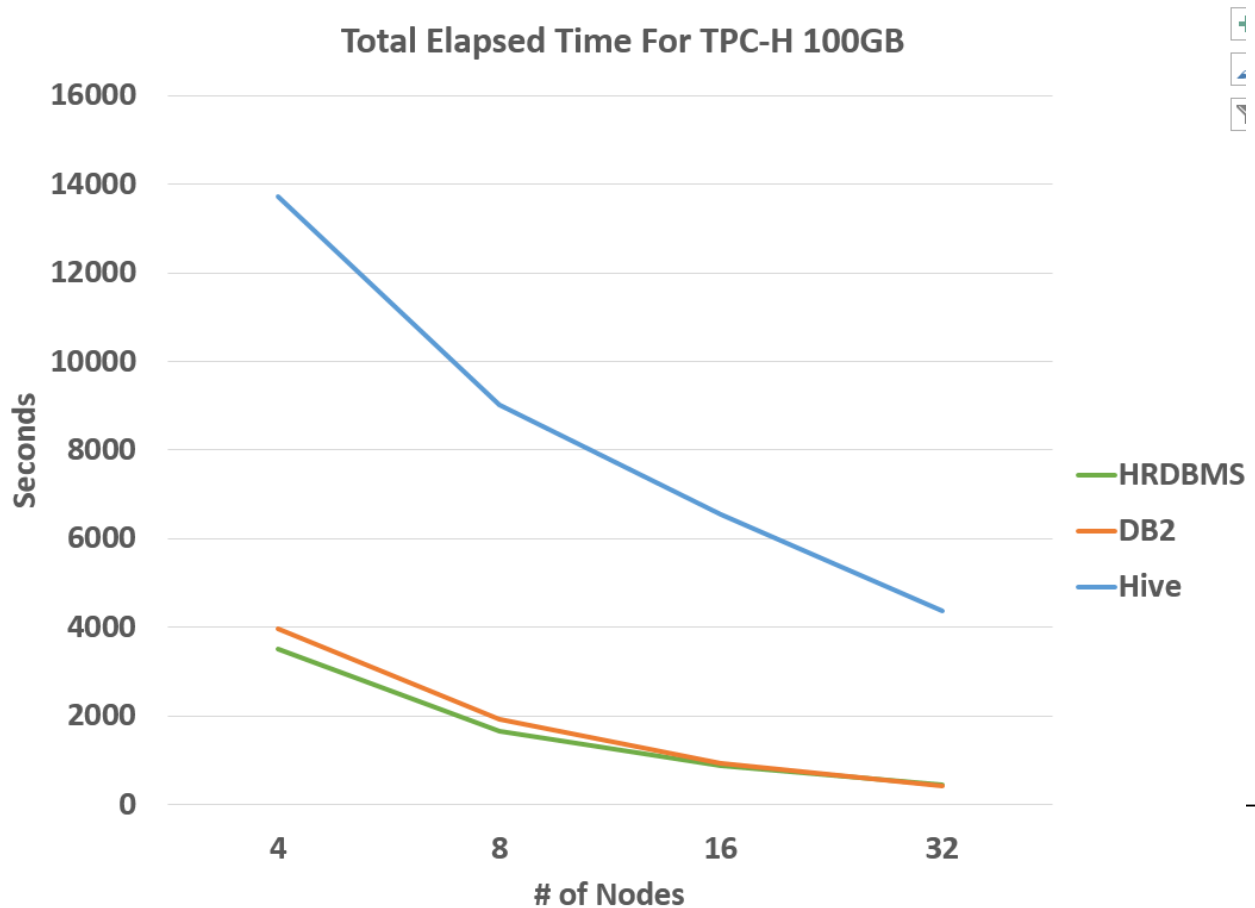
- Primary goals
  - The HRDBMS execution engine is tailored for SQL analytics workloads
  - Many novel features have been implemented
    - Predicate cache
    - Bloom filters for all join types
    - Buffering/caching services for the MapReduce engine
    - Map phase combination
    - Network data flow considered as part of query optimization
    - First MapReduce-like database with full transaction isolation and consistency
- **HRDBMS is a fully functional database**
  - Written from scratch - ~150k lines of code
  - 28 months of work
  - Written entirely by myself

How Does HRDBMS Accomplish These Goals?

- Secondary goals
  - HRDBMS does not use HDFS
  - So it becomes trivial to implement update and delete operations
  - It uses the local filesystem of each worker node in the cluster
  - The coordinator node keeps track of how data is spread across nodes and disks
  - Standard SQL support from the beginning
  - All operations can run externally if they won't fit in memory

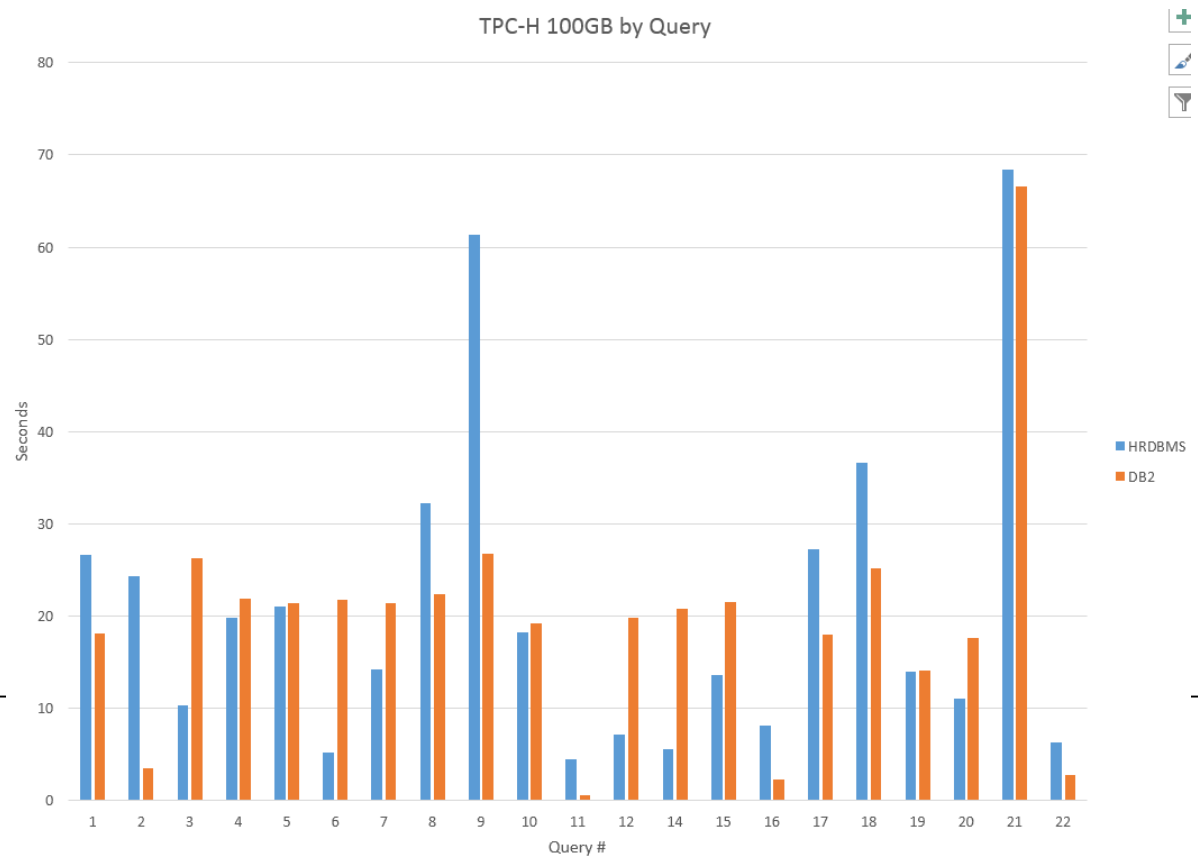
Results

- Testing shows that we can achieve per-node performance on par with traditional MPP relational databases while still using a MapReduce-like computation model



Results

- Testing was done with the TPC-H benchmark
  - Industry standard benchmark testing many different aspects of large scale analytics performance
  - HRDBMS outperforms a traditional MPP relational database on many queries and struggles on a few other queries



Conclusions

- Can we eliminate the framework bottlenecks in databases like Hive and Spark SQL to achieve better per-node performance?
  - The results show us that yes we can
- Does the custom HRDBMS framework still scale well like traditional MapReduce?
  - Future work is required to demonstrate this

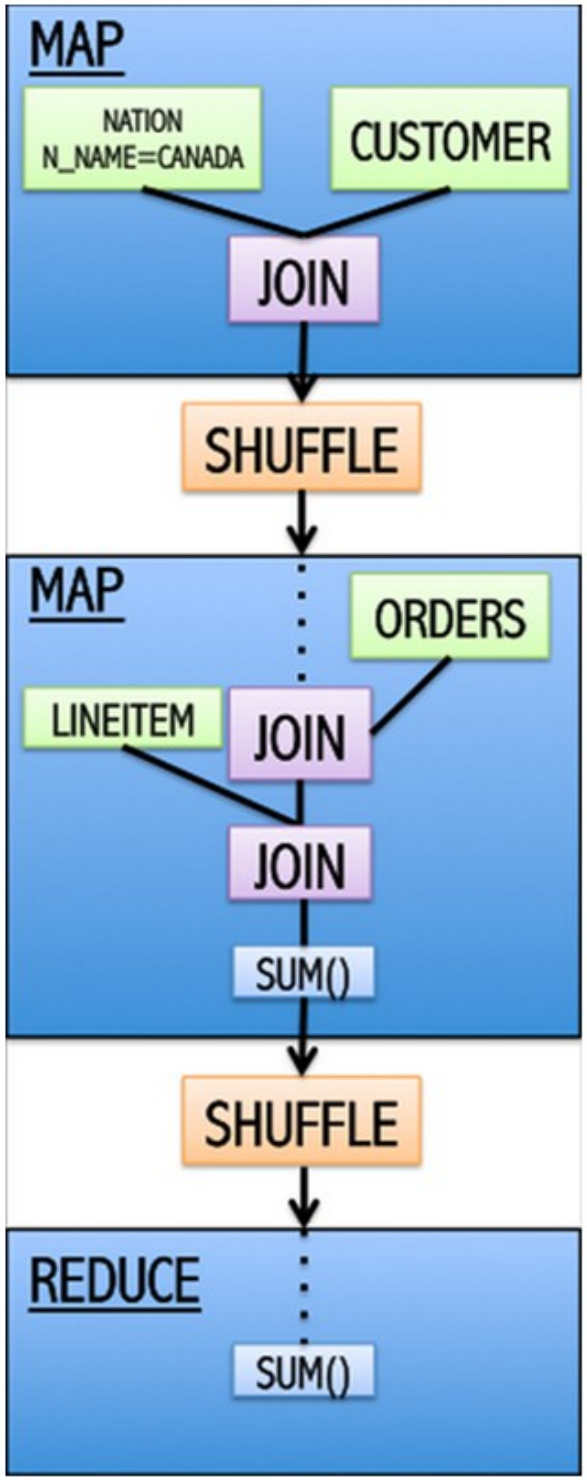
Future Work

- Larger scale testing
- HRDBMS does not yet have fault tolerance, which is required for applications designed to run on large clusters
  - We have a basic design for how fault tolerance will be implemented
- I intend to do some further research around removing the need for coordinator nodes
  - Worker nodes would be able to communicate directly back to the client in parallel
  - This would be novel work and I expect it to improve performance further

Solving Some Secondary Problems Along the Way

- Hive and Spark also have some other problems that present challenges for large data warehouse environments
  - New data can only be inserted (no UPDATEs or DELETEs)
  - In reality, DELETEs are usually not needed, but UPDATES are
- Hive (and other modern databases) do not support standard SQL and therefore can't be used with the plethora of existing SQL tooling that is available
  - Spark SQL recently added SQL support, but it still needs further work
- Certain operations in Hive and Spark are expected to fit in memory
  - This can cause some SQL statements to fail even though they could have been executed if the operation could have run externally

	Supports Large Clusters	Full DML	Full ACID	Good performance data > mem	Standard syntax for basic SQL	Commodity Hardware
DB2		X	X	X	X	X
Netezza		X	X	X	X	
Teradata		X	X	X	X	
Greenplum		X	X	X	X	X
HANA		X	X		X	X
Hive	X			X		X
Spark	X				X	X


MAP
NATION
N_NAME=CANADA
CUSTOMER
JOIN
SHUFFLE
MAP
ORDERS
LINEITEM
JOIN
JOIN
SUM()
SHUFFLE
REDUCE
SUM()

Predicate Caching

- Assume we need rows where column A > 5
  - We know nothing about this predicate, except that using an index won't be efficient
  - We scan the table to find the matching rows
  - During the scan we learn that pages 5, 12, and 17 have no rows that qualify
  - We cache this knowledge
- Now we get a query where we need A > 10
  - We automatically know that we don't need to read pages 5, 12, or 17
- Assume that something on page 12 changes
- Now we get a query with A > 7
  - We know that we don't need to read pages 5 or 17
- Query comes in with A > 4
  - We have to read everything