Shark: SQL & Rich Analytics at Scale

Deborah Hanus & Victor Lei

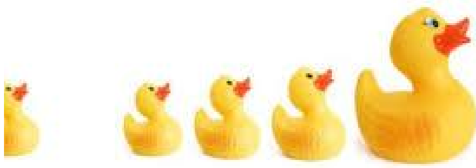
What problem does Shark solve?

What is the problem? Why is it hard?

Increasing data volumes

What is the problem? Why is it hard?

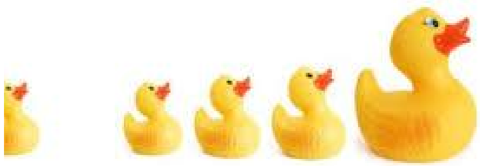
Increasing data volumes



Increased incidence of stragglers & faults

What is the problem? Why is it hard?

Increasing data volumes

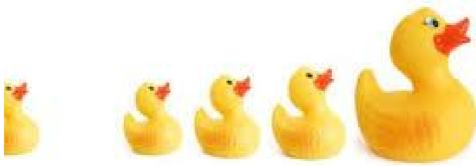


Increased incidence of stragglers & faults

Data analysis more difficult

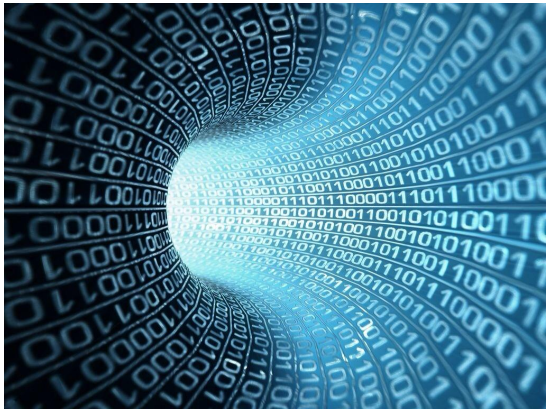
What is the problem? Why is it hard?

Increasing data volumes



Increased incidence of stragglers & faults

Data analysis more difficult



Users want to query at interactive speeds

Existing solutions

Queries to MapReduce (eg. Hive, Cheetah)

Highly scalable

Fine-grained fault tolerance

High latency

Existing solutions

Queries to MapReduce (eg. Hive, Cheetah)

Highly scalable

Fine-grained fault tolerance

High latency

Shared-nothing parallel DBs (eg. Impala, PowerDrill)

Less scalable

No fine-grained fault tolerance

Low latency

Existing solutions

Queries to MapReduce (eg. Hive, Cheetah)

Highly scalable

Fine-grained fault tolerance

High latency

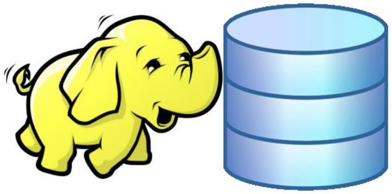
Shared-nothing parallel DBs (eg. Impala, PowerDrill)

Less scalable

No fine-grained fault tolerance

Low latency

Hybrid approaches (eg. HadoopDB, Osprey)



What is the core intuition of Shark's solution?

What is the core intuition?

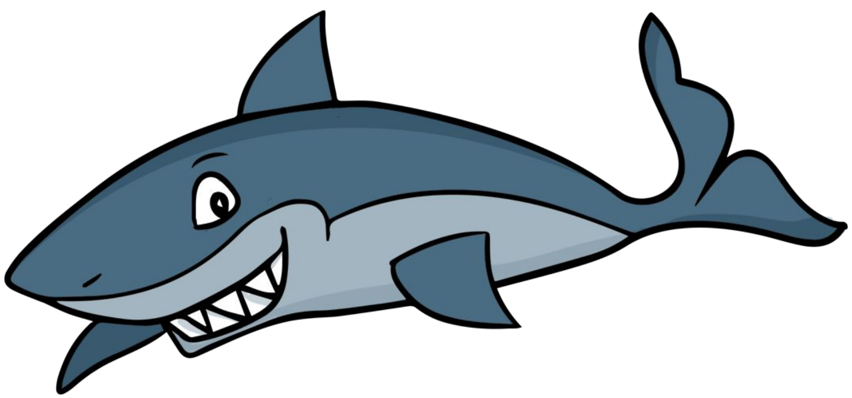
RDDs are the best.

Modern Databases are OK too.

What is the core intuition?

RDDs are the best.

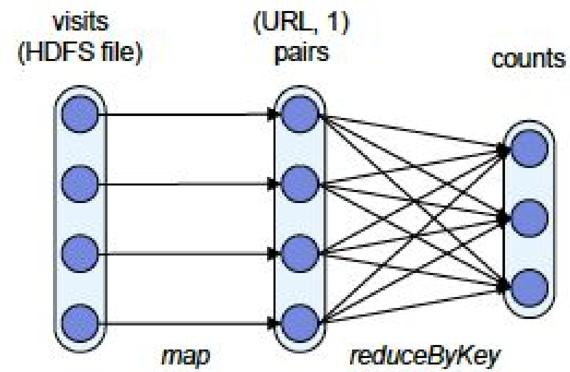
Modern Databases are OK too.



Shark: Scalable, Fault Tolerant, Interactive query speeds

Solution: Resilient Distributed Datasets (RDDs)

Read-only partitioned collection of records stored in-memory.



Representation:

- Set of partitions – atomic pieces of the data set
- Set of dependencies on parent RDDs
- Function to compute dataset based on its parents
- Metadata about partitioning scheme and data placement

Created by *transformations* of (a) data in stable storage, or (b) other RDDs

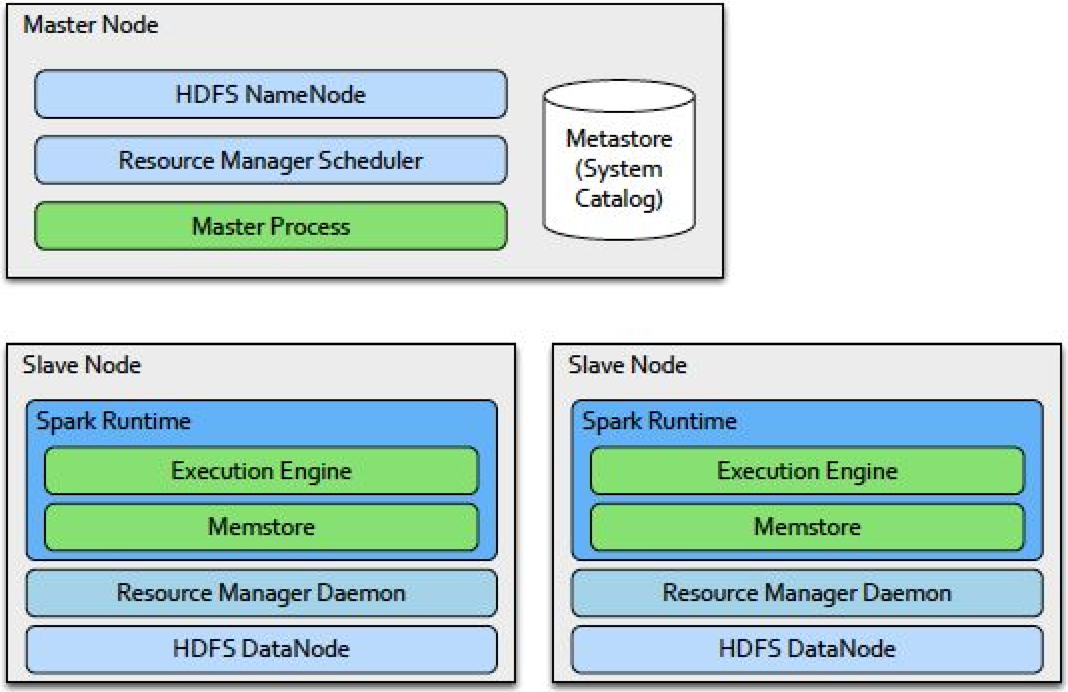
Coarse-grained writes make it fault tolerant.

How do RDDs compare to standard distributed shared memory systems?

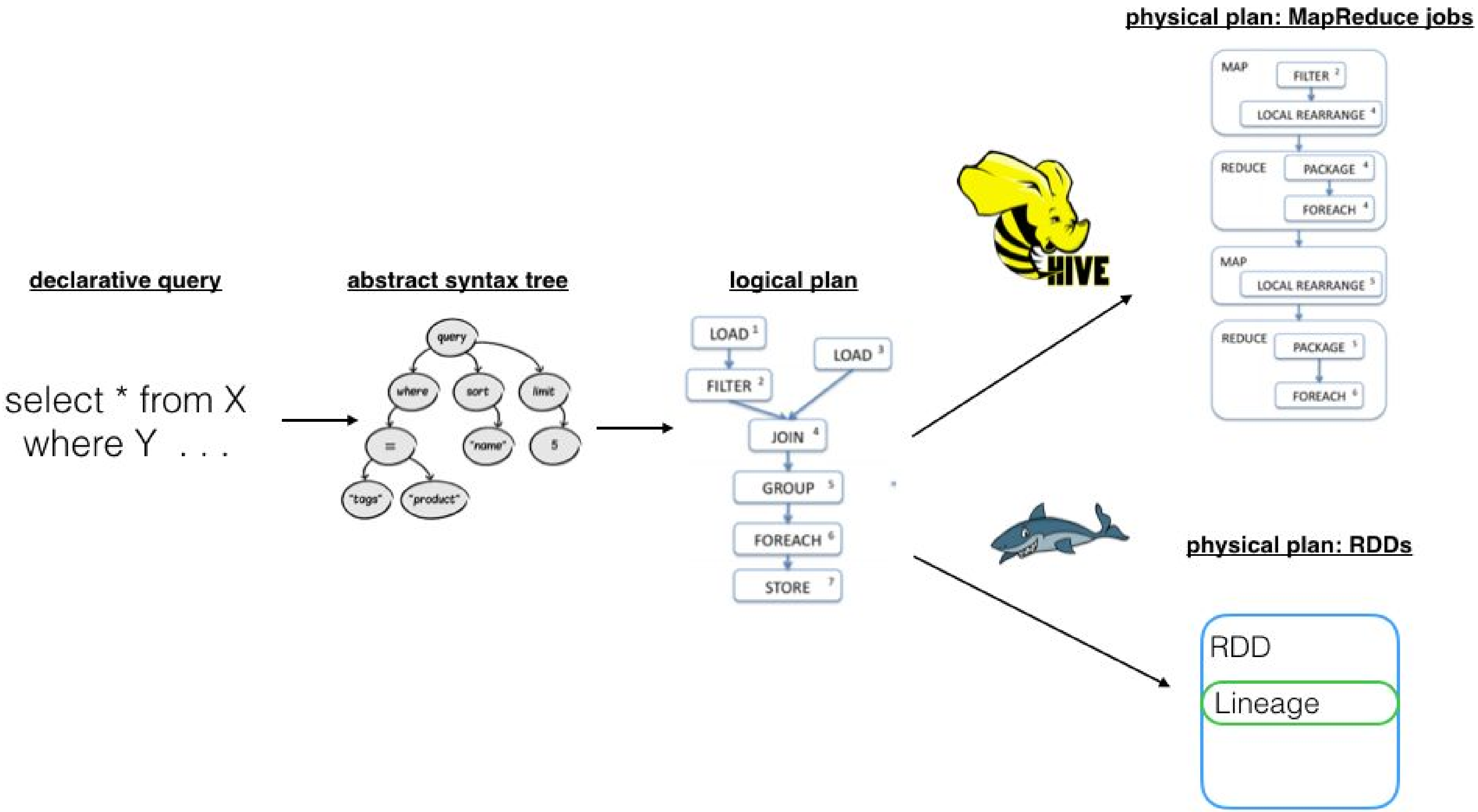
RDD's vs. Distributed Shared Memory Systems

Aspect	RDDs	Distr. Shared Mem.
Reads	Coarse- or fine-grained	Fine-grained
Writes	Coarse-grained	Fine-grained
Consistency	Trivial (immutable)	Up to app / runtime
Fault recovery	Fine-grained and low-overhead using lineage	Requires checkpoints and program rollback
Straggler mitigation	Possible using backup tasks	Difficult
Work placement	Automatic based on data locality	Up to app (runtimes aim for transparency)
Behavior if not enough RAM	Similar to existing data flow systems	Poor performance (swapping?)

Solution: Shark Architecture



Solution: Executing SQL on RDDs



Solution: Engine Extensions

1. **Partial DAG Execution**

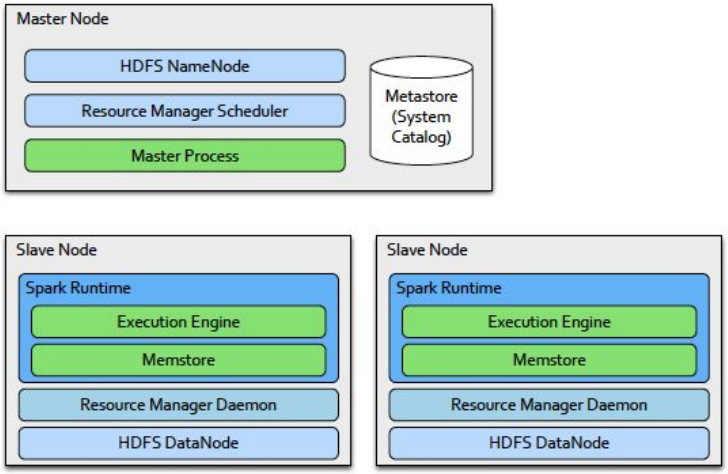
Gathers statistics at global & partition granularities

Allows DAG to be altered based on stats

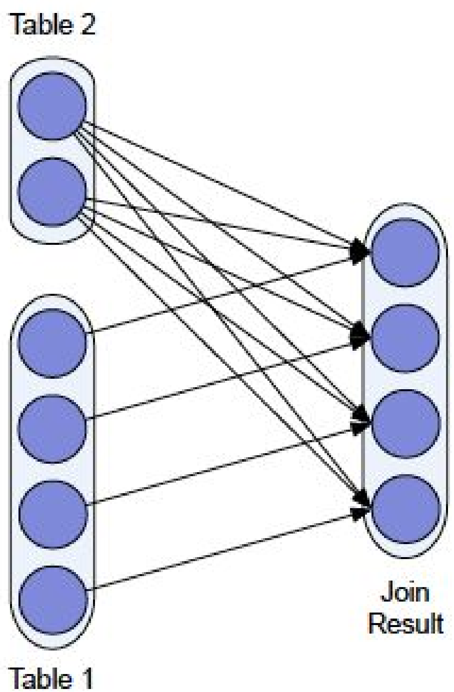
Worker sends stats to master, which optimizes

Handles skew and degree of parallelism

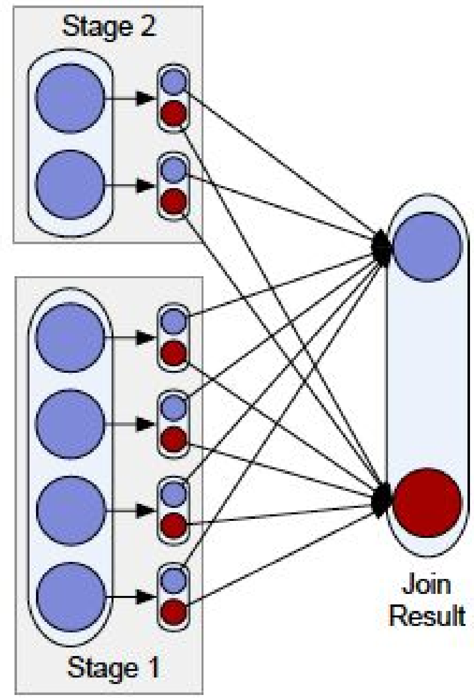
Allows join optimization



Solution: Partial DAG execution & join optimization



Map join



Shuffle join

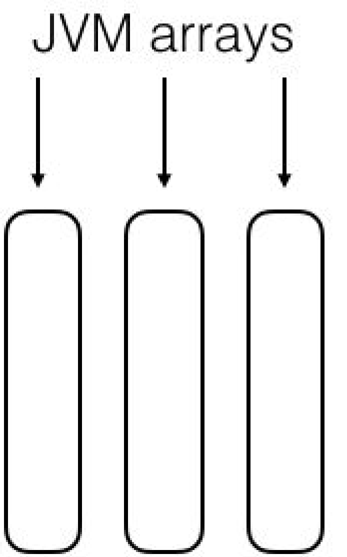
Solution: Engine Extensions

2. Columnar Memory Store

Placing objects in memory increases speed

All columns are JVM arrays

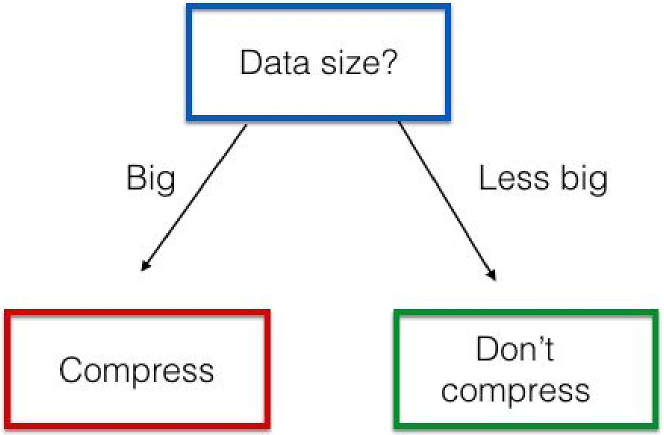
Each column creates only one JVM object



Solution: Engine Extensions

3. Distributed Data Loading

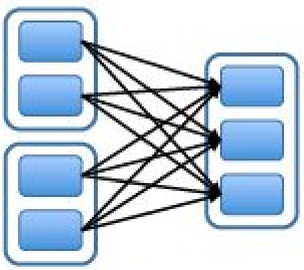
Tracks metadata for each task, determining if it should be compressed



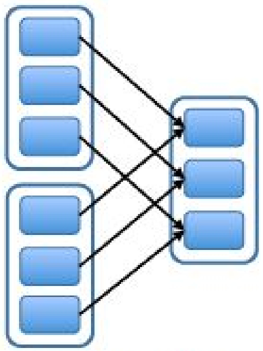
Solution: Engine Extensions

4. Data Co-partitioning

If you know the data schema, you can avoid shuffles by co-partitioning



join with inputs not co-partitioned

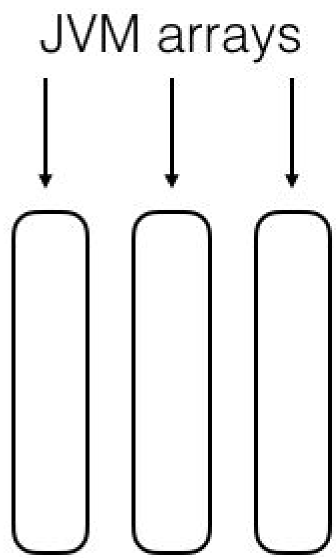


join with inputs co-partitioned

Solution: Engine Extensions

5. Partition Statistics & Map Pruning

Shark can avoid blocks that only contain data outside of the query range



How does this compare to modern database systems?

How does this compare to modern databases?

Cache conscious

Adaptive query patterns (e.g. H2O)

Parallelism for minimal lock contention

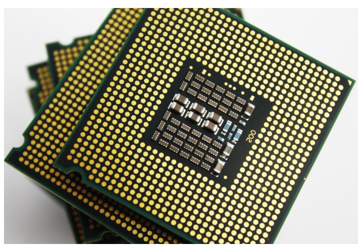
Shared nothing architecture (e.g. SharedDB)

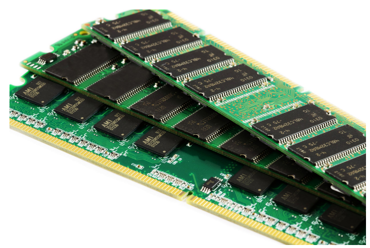
Experiments

1. Pavlo et al. experiments (2.1 TB dataset)
2. TPC-H Dataset (100GB and 1TB)
3. Sampled real workload from Shark user (1.7TB)
4. Synthetic Machine Learning Dataset (100GB)

Experiments

amazon web services EC2 x 100





8 cores

68 GB

1.7 TB

Results

Shark can perform 100x faster than Hive and Hadoop

Even faster* than MPP databases in some experiments

How does this compare to other machine learning systems?

Advantages of Shark for Machine Learning

Keeping data in memory (RDDs)

Machine Learning as a first-class citizen: incorporating SQL with Machine Learning

Machine Learning

1 B rows

10 columns

100 GB data

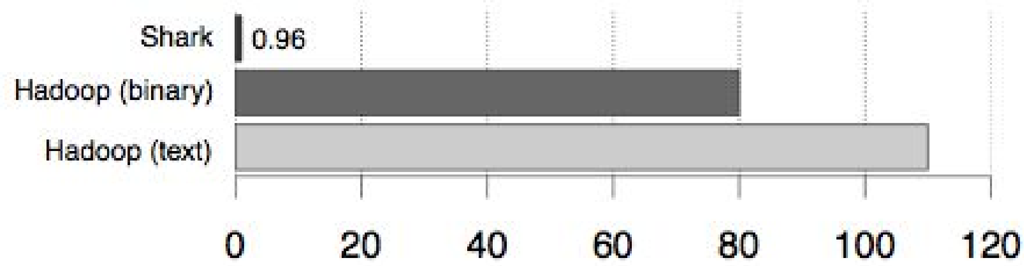


Figure 10: Logistic regression, per-iteration runtime (seconds)

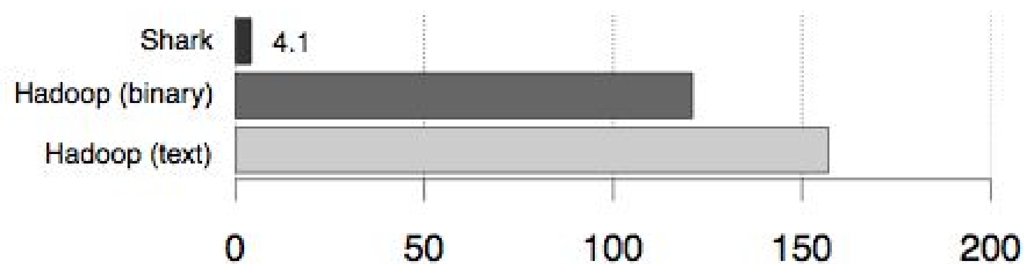


Figure 11: K-means clustering, per-iteration runtime (seconds)

Aggregation performance

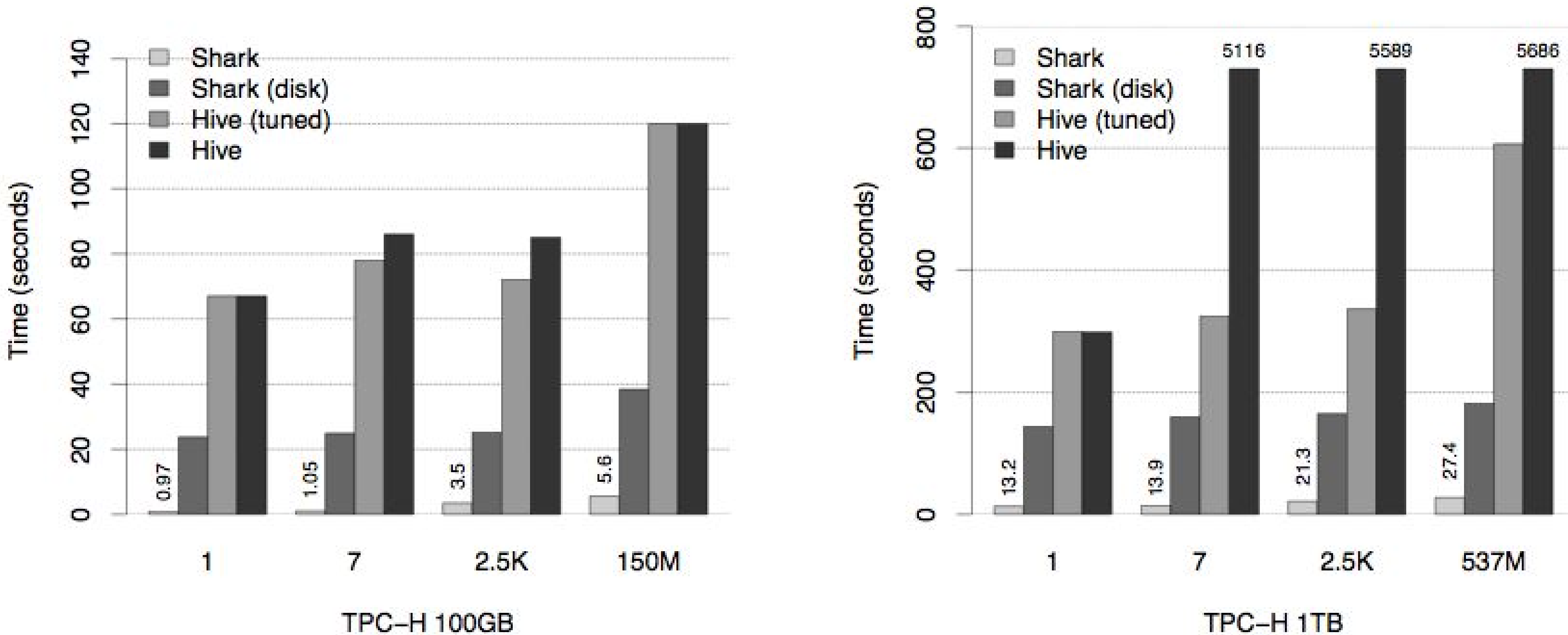


Figure 6: Aggregation queries on *lineitem* table. X-axis indicates the number of groups for each aggregation query.

```
SELECT [GROUP_BY_COLUMN], COUNT(*) FROM lineitem GROUP BY
[GROUP_BY_COLUMN]
```

Join strategy optimization

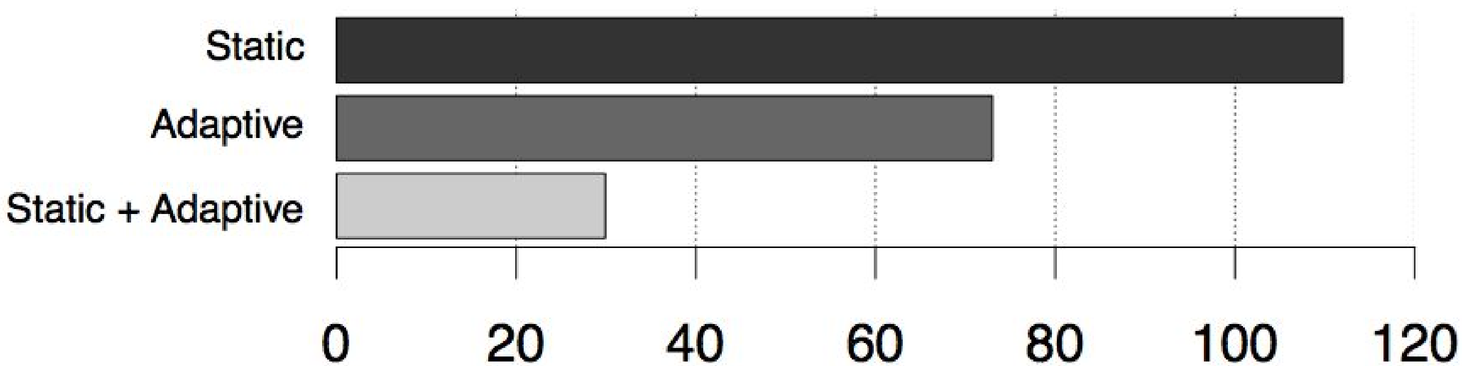


Figure 7: Join strategies chosen by optimizers (seconds)

A combination of static query analysis and partial DAG execution led to a 3x performance improvement over a naïve, statically chosen plan

Real Hive Warehouse Queries

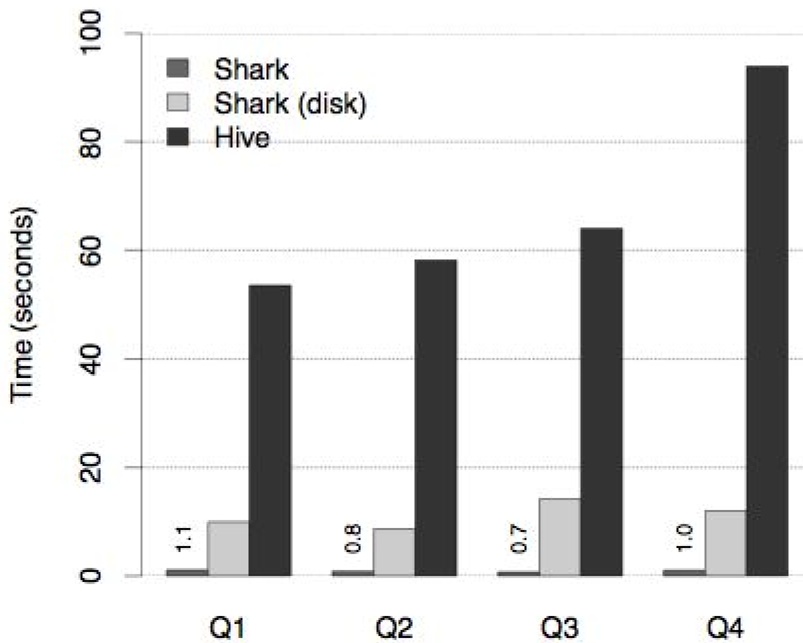


Figure 9: Real Hive warehouse workloads

The map pruning technique, on average, reduced the amount of data scanned by <u>a factor of 30</u>

What else could be done with this paper?

Next steps

Add the two unimplemented areas (bytecode compilation and specialized data structures) and benchmark differences in performance

More extensive real-world machine learning benchmarks (including machine learning tools)

Looking at other machine learning algorithms that can be incorporated into Shark