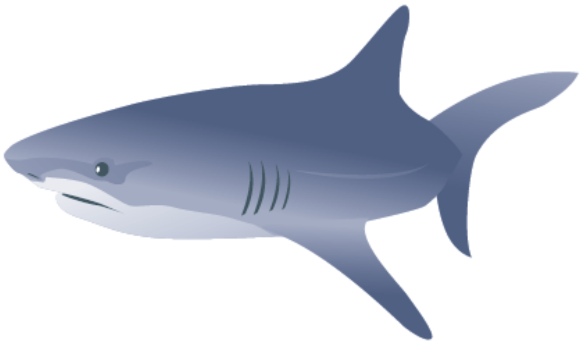
Shark

Hive on Spark



Cliff Engle, Antonio Lupher, Reynold Xin,
Matei Zaharia, Michael Franklin, Ion Stoica,
Scott Shenker

Agenda

- Intro to Spark
- Apache Hive
- Shark
- Shark’s Improvements over Hive
- Demo
- Alpha status
- Future directions

What Spark Is

- *Not* a wrapper around Hadoop
- Separate, *fast*, MapReduce-like engine
  - In-memory data storage for very fast iterative queries
  - Powerful optimization and scheduling
  - Interactive use from Scala interpreter
- Compatible with Hadoop's storage APIs
  - Can read/write to any Hadoop-supported system, including HDFS, HBase, SequenceFiles, etc

Project History

- Started in summer of 2009
- Open sourced in early 2010
- In use at UC Berkeley, UCSF, Princeton, Klout, Conviva, Quantifind, Yahoo! Research

Spark Programming Model

Resilient distributed datasets (RDDs)

Distributed collections of Scala objects

Can be cached in memory across cluster nodes

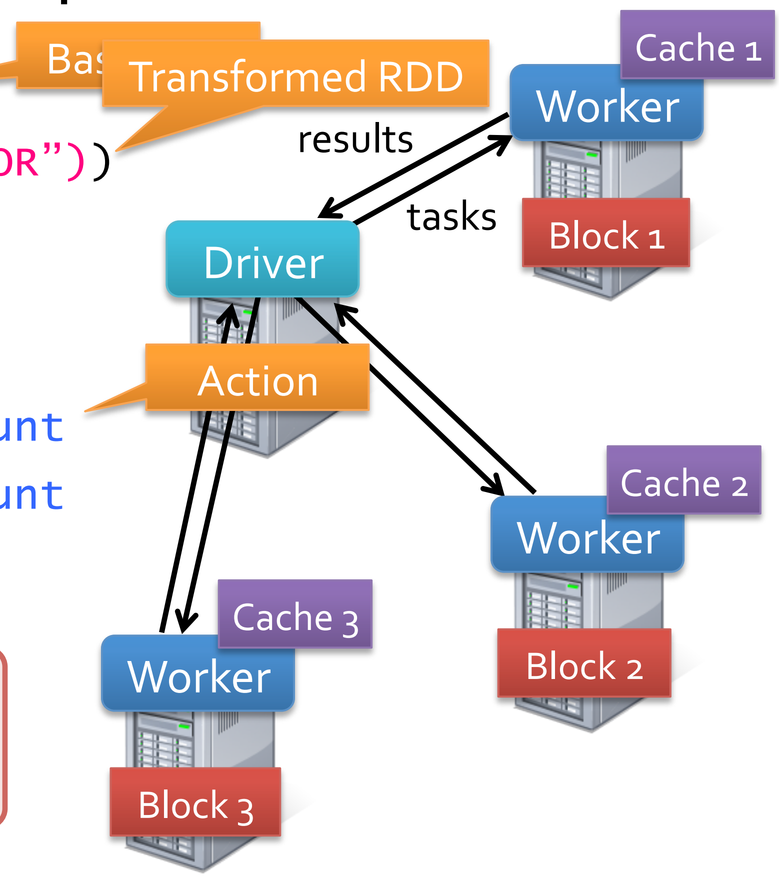
- Manipulated like local Scala collections

Automatically rebuilt on failure

Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

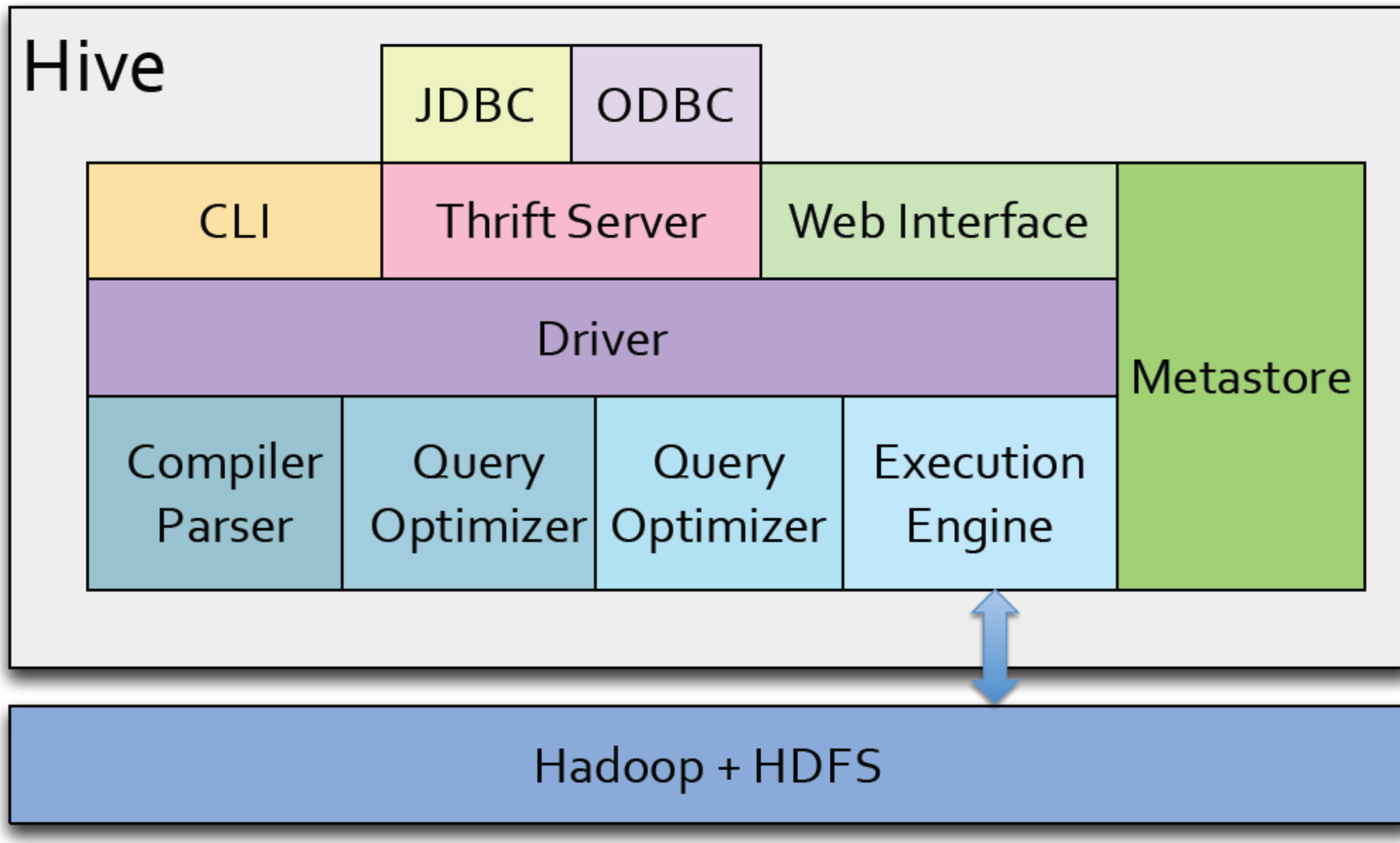
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
. . .
```



Result: scaled to 1 TB data in 5-7 sec
(vs 170 sec for on-disk data)

Apache Hive

- Data warehouse solution developed at Facebook
- SQL-like language called HiveQL to query structured data stored in HDFS
- Queries compile to Hadoop MapReduce jobs


Hive
JDBC
ODBC
CLI
Thrift Server
Web Interface
Metastore
Driver
Compiler Parser
Query Optimizer
Query Optimizer
Execution Engine
Hadoop + HDFS

Hive Principles

- SQL provides a familiar interface for users
- Extensible types, functions, and storage formats
- Horizontally scalable with high performance on large datasets

Hive Applications

- Reporting
- Ad hoc analysis
- ETL for machine learning

…

Hive Downsides

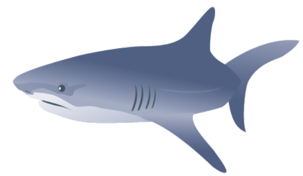
- Not interactive
  - Hadoop startup latency is ~20 seconds, even for small jobs
- No query locality
  - If queries operate on the same subset of data, they still run from scratch
  - Reading data from disk is often bottleneck
- Requires separate machine learning dataflow

Shark Motivation

- Exploit temporal locality: working set of data can often fit in memory to be reused between queries
- Provide low latency on small queries
- Integrate distributed UDF’s into SQL

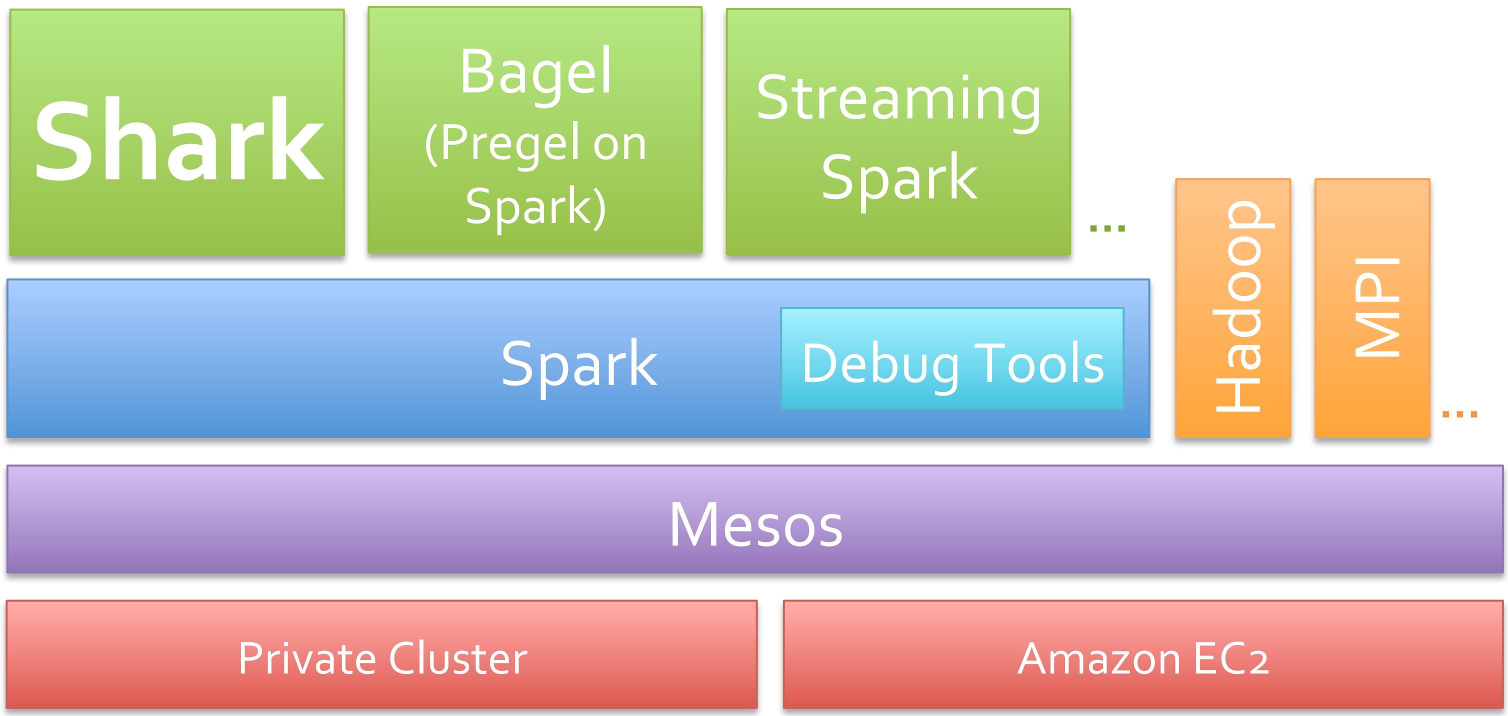
Introducing Shark

- Shark = Spark + Hive



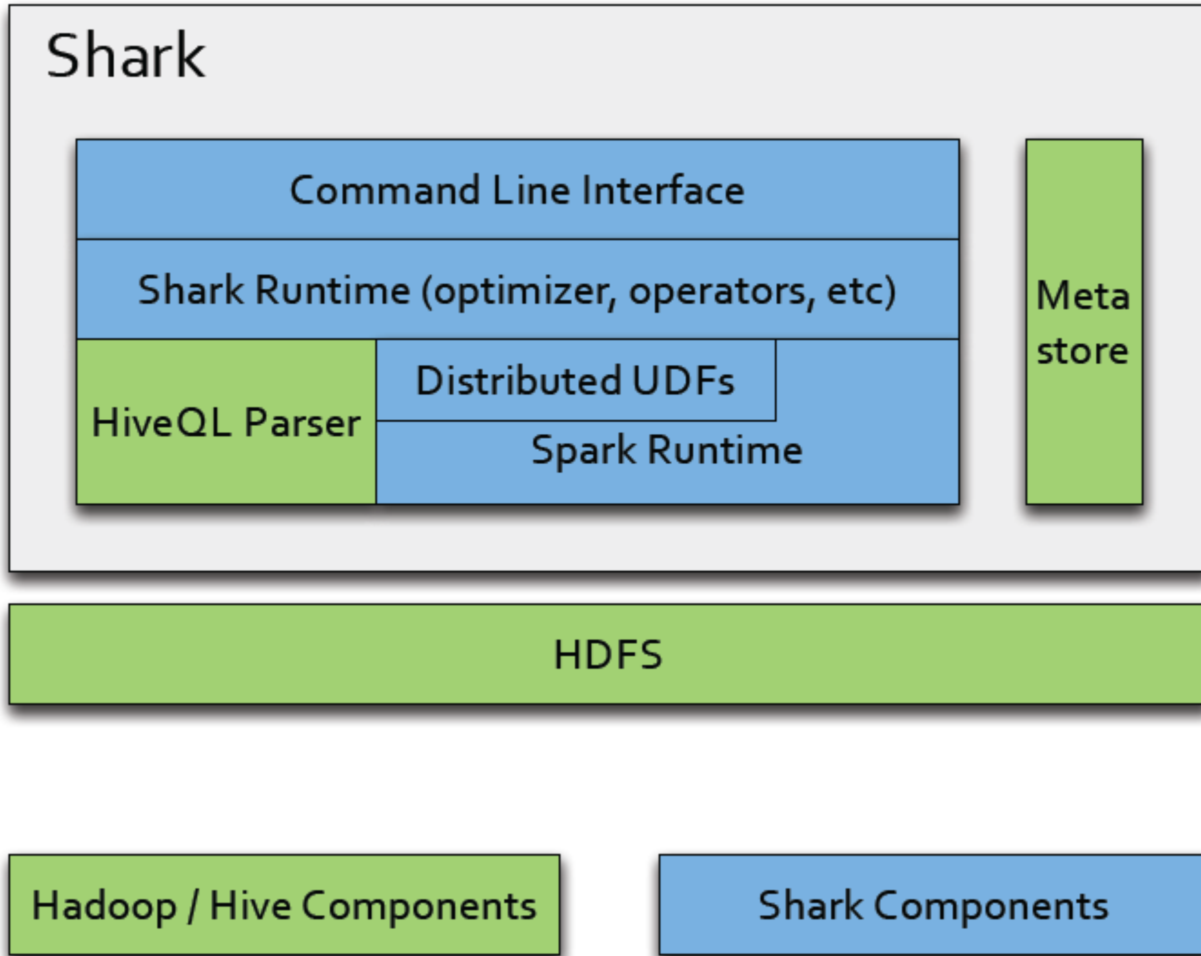
- Run HiveQL queries through Spark with Hive UDF, UDAF, SerDe
- Utilize Spark's in-memory RDD caching and flexible language capabilities

Shark in the AMP Stack



Shark

- Physical operators using Spark
- Rely on Spark's fast execution, fault tolerance, and in-memory RDD's
- Reuse as much Hive code as possible
  - Convert logical query plan generated from Hive into Spark execution graph
- Compatible with Hive
  - Run HiveQL queries on existing HDFS data using Hive metadata, **without modifications**


Shark
Command Line Interface
Shark Runtime (optimizer, operators, etc)
Meta store
HiveQL Parser
Distributed UDFs
Spark Runtime
HDFS
Hadoop / Hive Components
Shark Components

0.1 alpha: 84% Hive tests passing (575 out of 683)

http://github.com/amplab/shark

0.1 alpha

- Experimental SQL/RDD integration
- User selected caching with CTAS
- Columnar RDD cache
- Some performance improvements

Caching

- User selected caching with CTAS

- CREATE TABLE mytable_cached AS SELECT * FROM mytable WHERE count > 10;

- mytable_cached becomes an in-memory materialized view that can be used to answer queries.

SQL/Spark Integration

- Allow users to implement sophisticated algorithms as UDFs in Spark
- Query processing UDFs are streamlined

```
val rdd = sc.sql2rdd(
  "select foo, count(*) c from pokes group by foo")

println(rdd.count)

println(rdd.mapRows(_.getInt("c")).reduce(_+_))
```

Performance optimizations

- Hash-based shuffle (speeds up group-bys)
  - Hadoop does sort-based shuffle which can be expensive.
- Limit push-down in order by
  - select * from pokes order by foo limit 10
- Columnar cache

Large Caches in JVM

- Java objects have large overhead (**2-4X** the actual data size)
- Boxing/Unboxing is slow
- GC is a bottleneck if we have many long-lived objects

Example: Caching Rows

Int, Double, Boolean

25	10.7	True

24 + 24 + 16 = **64 bytes***

Should be ~ **13 bytes**

*Estimated on 64 bit VM. Implementations may vary.

Possible Solutions

1. Store deserialized Java objects
2. Serialize objects to in-memory byte array
3. Use memory slab external to JVM
4. Store data in primitive arrays

Deserialized Java Objects

+ Fast (requires little CPU)

    (Up to 5X speedup)

- Poor memory usage (3X worse)

- Lots of Garbage Collection overhead

Serialize to Bytes

+ Best memory efficiency

+ Efficient GC

- CPU usage to serialize/deserialize each obj

Store in External Slab

+ Efficient memory

+ No GC

- CPU usage to serialize/ deserialize each obj

- More difficult to implement

Primitive Column Arrays

- + Efficient memory (similar to byte arrays)
- + Efficient GC
- + No extra CPU overhead

Column vs Row Storage

Columnar Benefits

- Better compression
- Fewer objects to track
- Only materialize necessary columns => Less CPU

Columnar in Shark

- Store all primitive columns in primitive arrays on each node
- Less deserialization overhead
- Space efficient and easier to garbage collect

Result

Achieved efficient storage space without sacrificing performance

Limit Pushdown in Sort

```
SELECT * FROM table
ORDER BY key LIMIT 10;
```

Main Idea

- The limit can be applied before sorting all data to increase performance.
- Each mapper computes its Top K and then a single reducer merges these

Possible Implementations

1. PriorityQueue (Heap) to keep running top k in one traversal on each mapper.

$$O(n \log k)$$

Possible Implementations

2. QuickSelect algorithm. Essentially quicksort, but prunes elements on one side of pivot when possible.

$O(n)$

Result

- We chose to use Google Guava’s implementation of QuickSelect.
- We observed much better performance than Hive which applies the limit after sorting on each reducer.

Demo

- 3-node EC2 large cluster (2 virtual cores, 7GB of RAM)

- Synthetic TPC-H data (2x scale factor)

Some numbers while we wait for Hive to finish running…

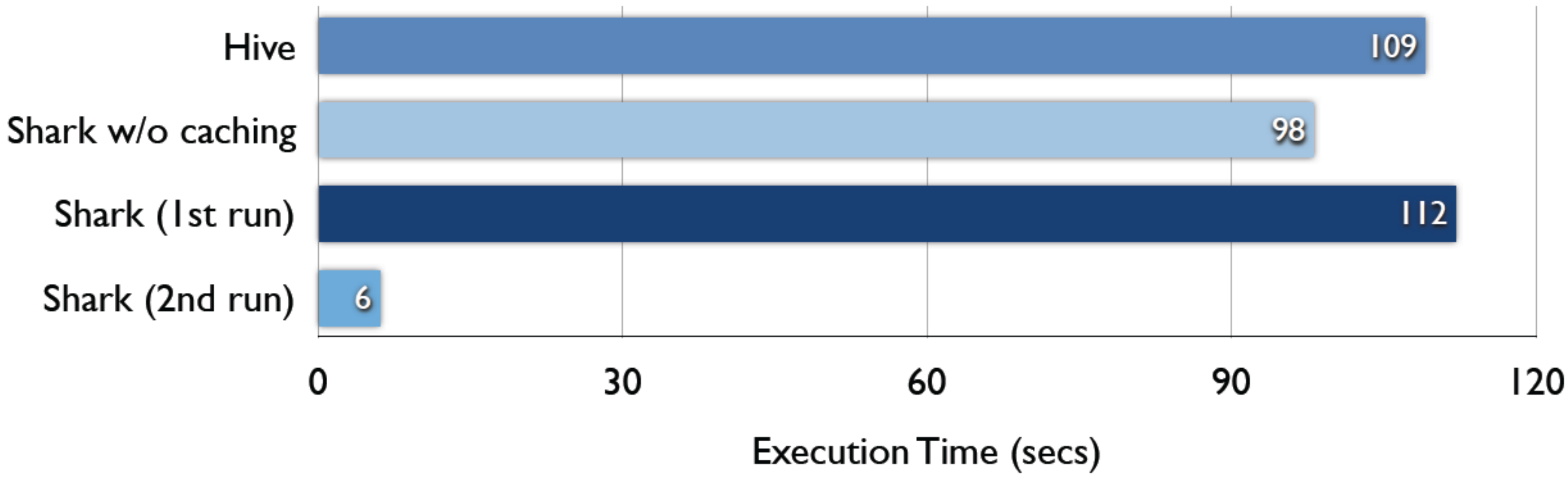
- Brown/Stonebraker benchmark 70GB[1]
  - Also used on Hive mailing list[2]
- 10 Amazon EC2 High Memory Nodes (30GB of RAM/node)
- Naively cache input tables
- Compare Shark to Hive 0.7

[1] http://database.cs.brown.edu/projects/mapreduce-vs-dbms/
[2] https://issues.apache.org/jira/browse/HIVE-396 [1]

Benchmarks: Query 1

- 30GB input table

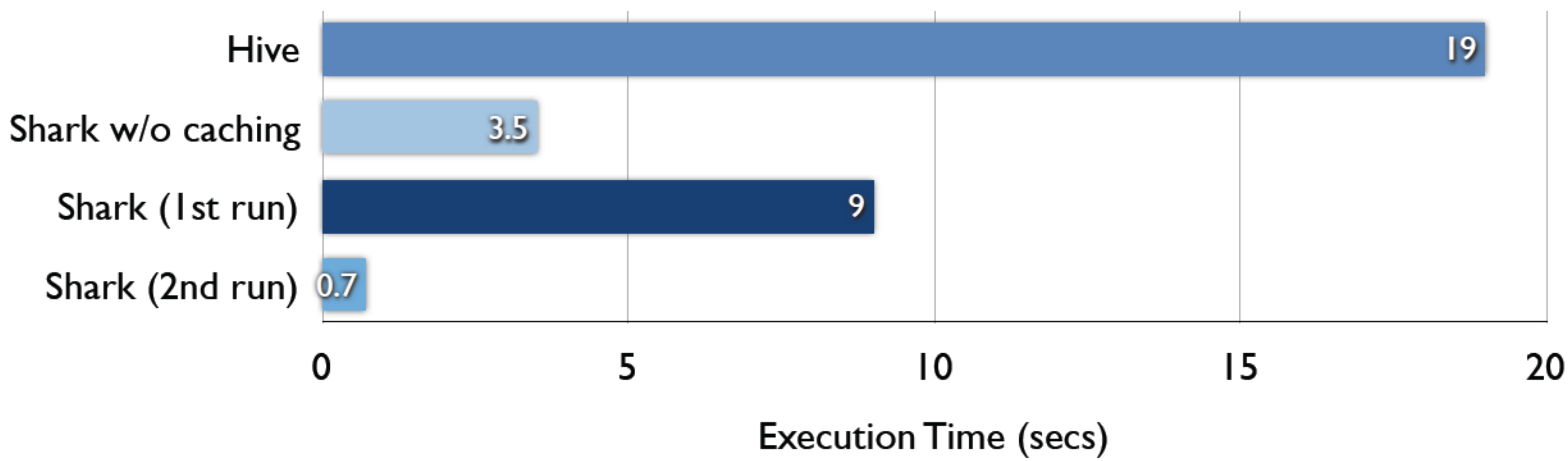
```
SELECT * FROM grep WHERE field LIKE '%XYZ%';
```



Benchmark: Query 2

- 5 GB input table

```
SELECT pagerank, pageURL FROM rankings WHERE
pagerank > 10;
```



Status

- Passing 575 Hive tests out of 683.

- You can help us decide what to implement next.

Unsupported Hive Features

- ADD FILE (use Hadoop distributed cache to distribute user-defined scripts)
- STREAMTABLE hint not supported
- Outer join with non-equi join filter
- Table statistics (piggyback file sink)
- Table with buckets
- ORDER BY with multiple reducers has a bug (will be fixed this week!)
- MapJoin has a serialization bug (will be fixed this week!)

Unsupported Hive Features

- Automatically convert joins to mapjoins
- Sort-merge mapjoins
- Partitions with different input formats
- Unique joins
- Writing to two or more tables in a single SELECT INSERT
- Archiving data
- virtual columns (INPUT__FILE__NAME, BLOCK__OFFSET__INSIDE__FILE)
- Merge small files from output

What's coming up (in a month)?

- Performance improvements (groupbys, joins)
- Demo of Shark on a 100-node cluster in SIGMOD 2012 (May 22, Scottsdale, AZ) mining Wikipedia visit log data

What's coming up (a year)?

- Shark-specific query optimizer
- Automatic tuning of parameters (e.g. number of reducers)
- Off-heap caching (e.g. EhCache)
- Automatic caching based on query analysis
- Shared caching infrastructure

Conclusion

- Shark is a warehouse solution compatible with Apache Hive
- Can be orders of magnitude faster
- We are looking for people to try and we are happy to provide support

Thank you!

Questions?

http://shark.cs.berkeley.edu

(will be updated tonight)