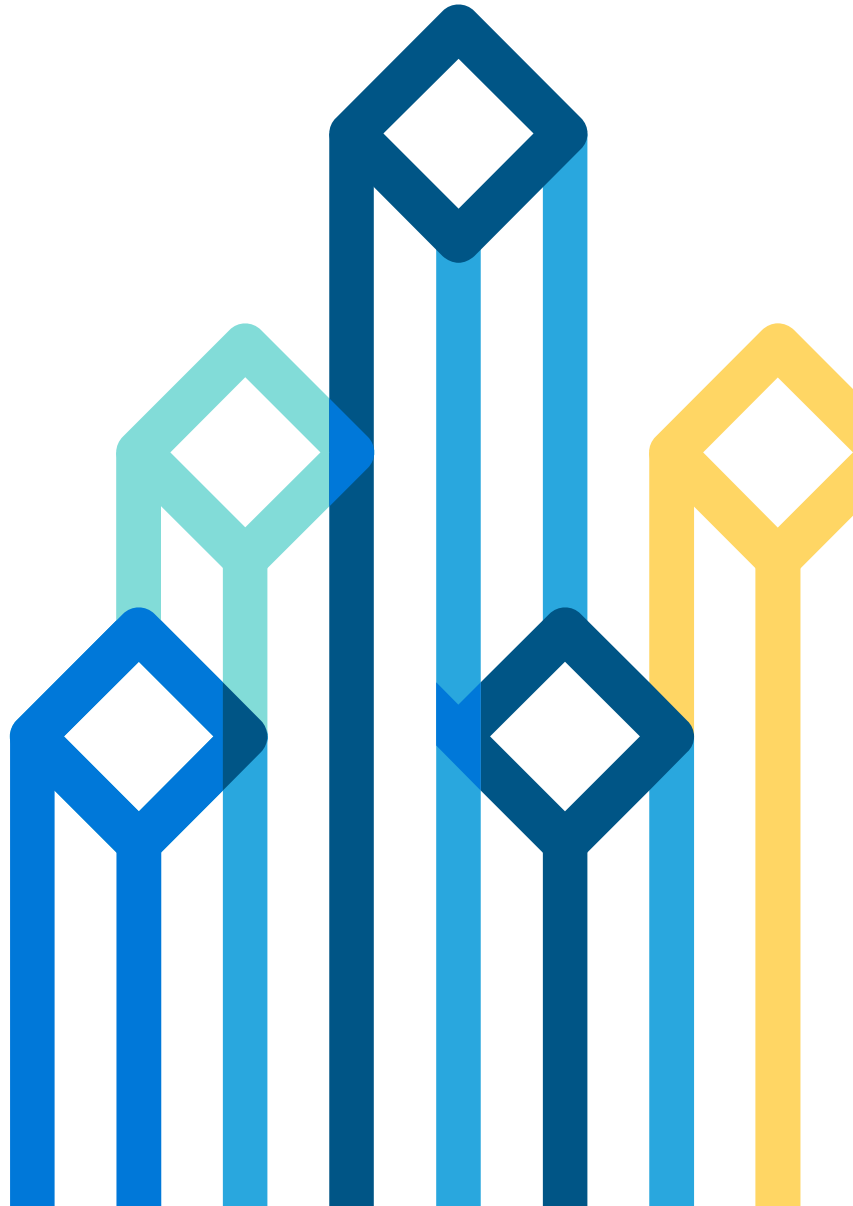
cloudera

Hive on Spark

Szehon Ho



My Background

- Cloudera:
  - Open-Source Distribution of Hadoop (CDH): Hadoop, Hbase, Hive, Impala, Kafka, Mahout, Oozie, Pig, Search, Spark, Zookeeper, many more
  - Enterprise Management and Security Tools

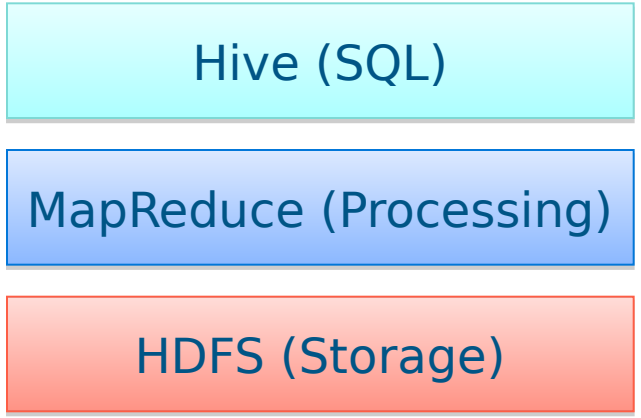
- Myself
  - Hive team member in Cloudera
  - Apache Hive Committer, PMC
  - Excited to be back in Germany

- **Background: Hive, Spark, Hive on Spark**
- Technical Deep Dive
- User-View

cloudera

Background: Hive

- MapReduce (2005)
  - Open-source distributed processing engine.
- Hive (2007)
  - Provides SQL access to MapReduce engine.
  - Main use-case in online analytic (data warehouse) space
  - Feature rich, mature (large community)
  - De-facto standard for SQL on Hadoop
  - Most-used Hadoop tool in Cloudera



Background: Spark

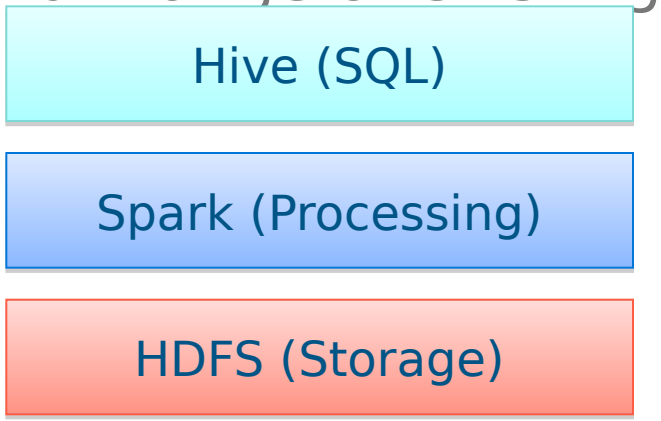
- Second wave of big-data innovation, many projects strive for improved distributed processing (Tez, Flink, etc)
- Spark (2009)
  - General consensus that its most well-placed to replace MapReduce.
  - Grown to be most active Apache project
  - Pig, Mahout, Cascading, Flume, Solr integrating or moving onto Spark.
  - Exposes more powerful API's and abstractions, very easy to use.

Background: Spark

	MapReduce	Spark
Data	File	RDD Kept in memory
Program	Map, Shuffle, Reduce In that order	Many more transformations Any order
Lifecycle	Tasks = Java Processes Short Lived Processes	Tasks != Java Process Long Lived Processes (Executors)

Hive on Spark: Goals

- Hive as access layer: Users can switch with minimal cost to better distributed processing engine => Better performance
- Goals:
  - Hive can run seamlessly on different processing engines (MR, Tez, and Spark).
  - Hive on Spark supports full range of existing Hive features



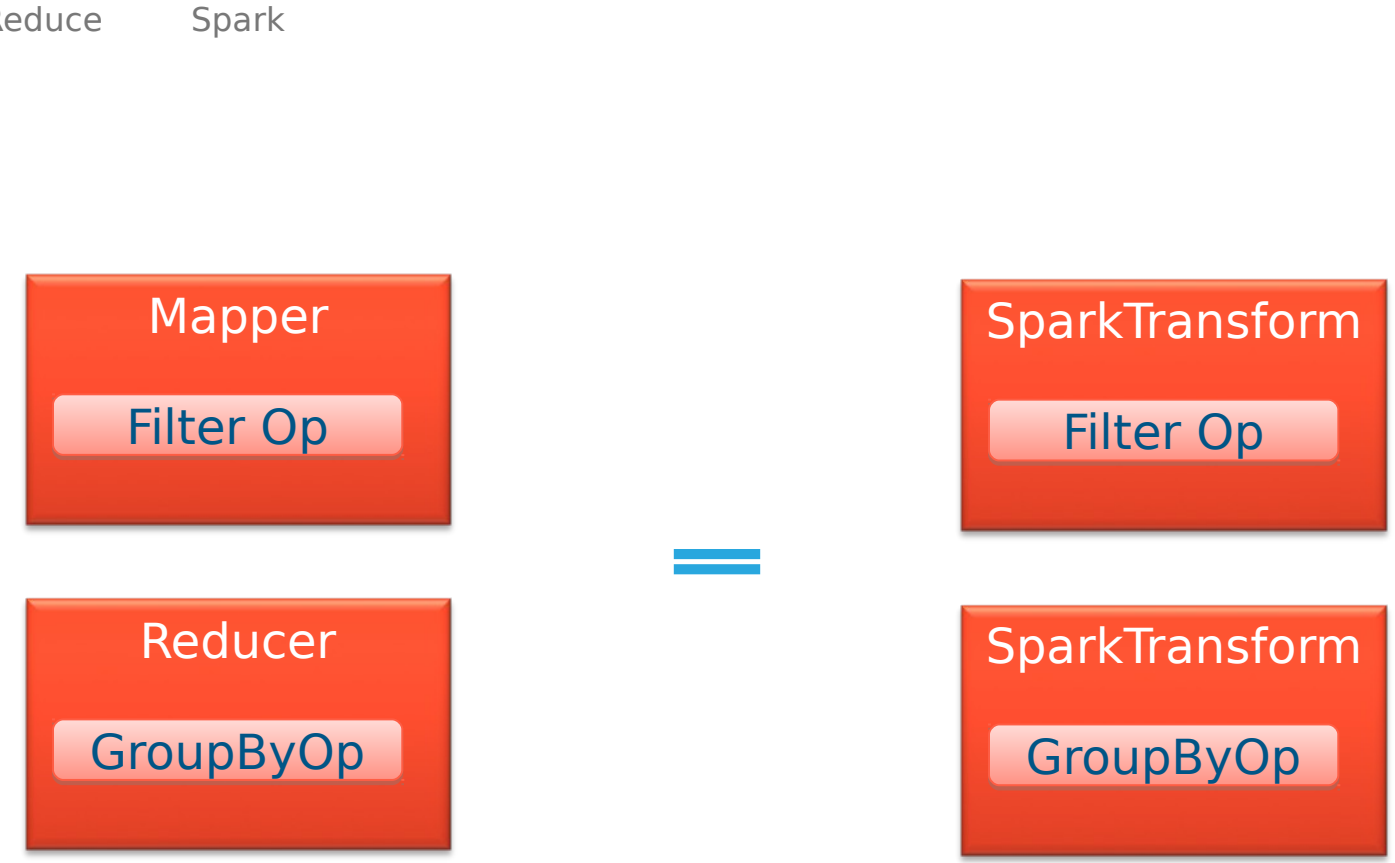
- Background: Hive, Spark, Hive on Spark
- **Technical Deep Dive**
- User-View

Design Concepts

- Challenge: Porting a mature system on a new processing engine
- Recap of advanced Functionality in Hive:
  - SQL Syntax
  - SQL data types
  - User-Defined Functions
  - File Formats
- Keeping most of the execution code (Hive operators) same across processing engines

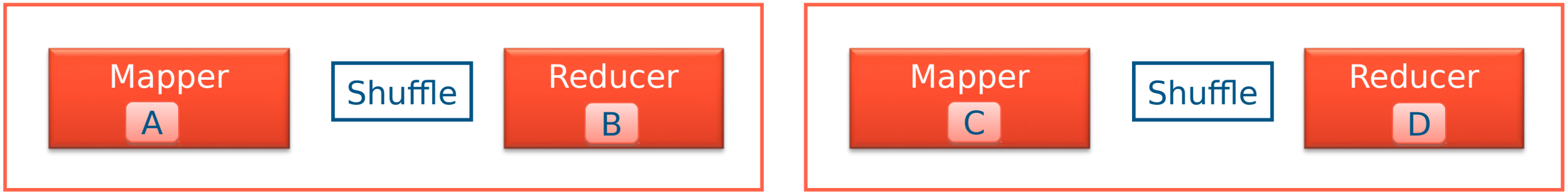
Design Concepts

- In general, we reuse the same Hive operators in Mapper/Reducer as in Spark local transformations.



Improvement: Eliminating Phases

- Spark allows us to organize same Hive operators in less phases
- MapReduce Job = Map Phase, Shuffle Phase, Red Phase

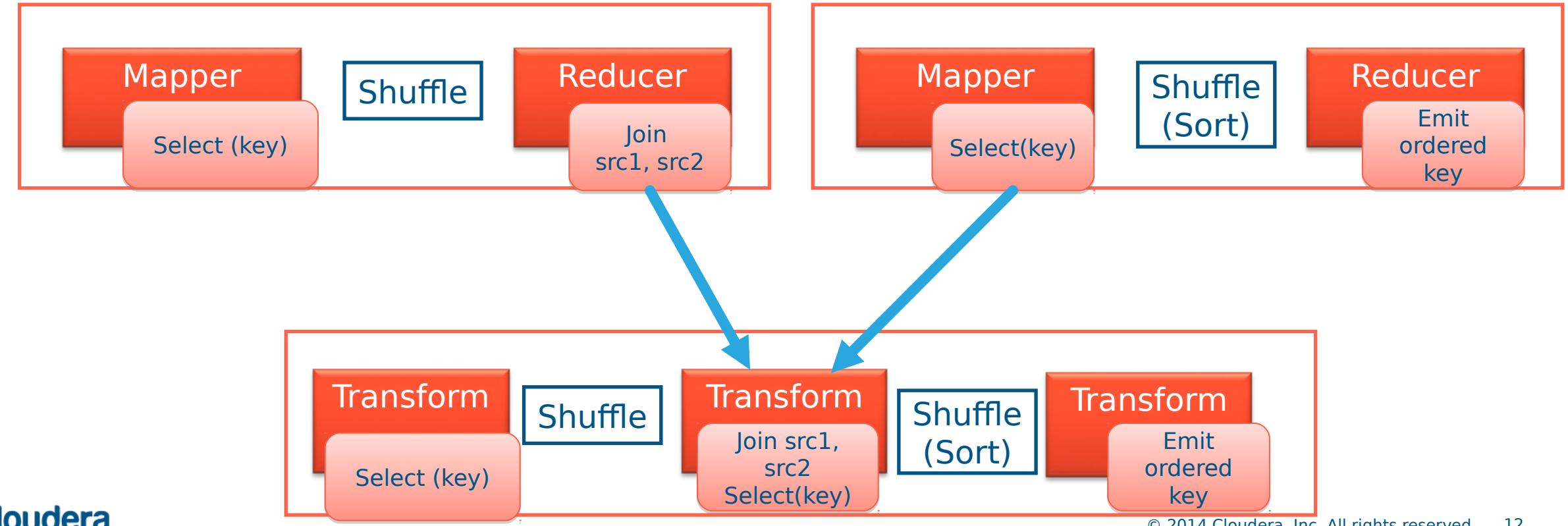


- Spark Job = Any number of “transformations” connected by ‘shuffles’

cloudera

Improvement: Eliminating Phases

```
SELECT src1.key FROM
   (SELECT key FROM src1 JOIN src2 ON src1.key = src2.key)
ORDER BY src1.key;
```



Improvement: In-Memory

- Files are input of Mapper, output of Reducer.
- More MapReduce jobs means more file IO (to temp Hive directory)

- The problem does not exist in Spark
- In-memory RDD as input/output of Spark transforms.

Improvement: Shuffle

- Shuffling is the bridge between Mapper and Reducer, it is data movement within one job.
- It is typically the most expensive part of MR job.
- Spark Shuffle: offers more efficient shuffling for specific use-cases

Improvement: Shuffle

- MapReduce shuffle-sort: hash-partitions and then sorts each partition.

- Select avg(value) from table group by key;
  - => Spark “groupBy” transform
  - In MapReduce, would do sorting unnecessarily

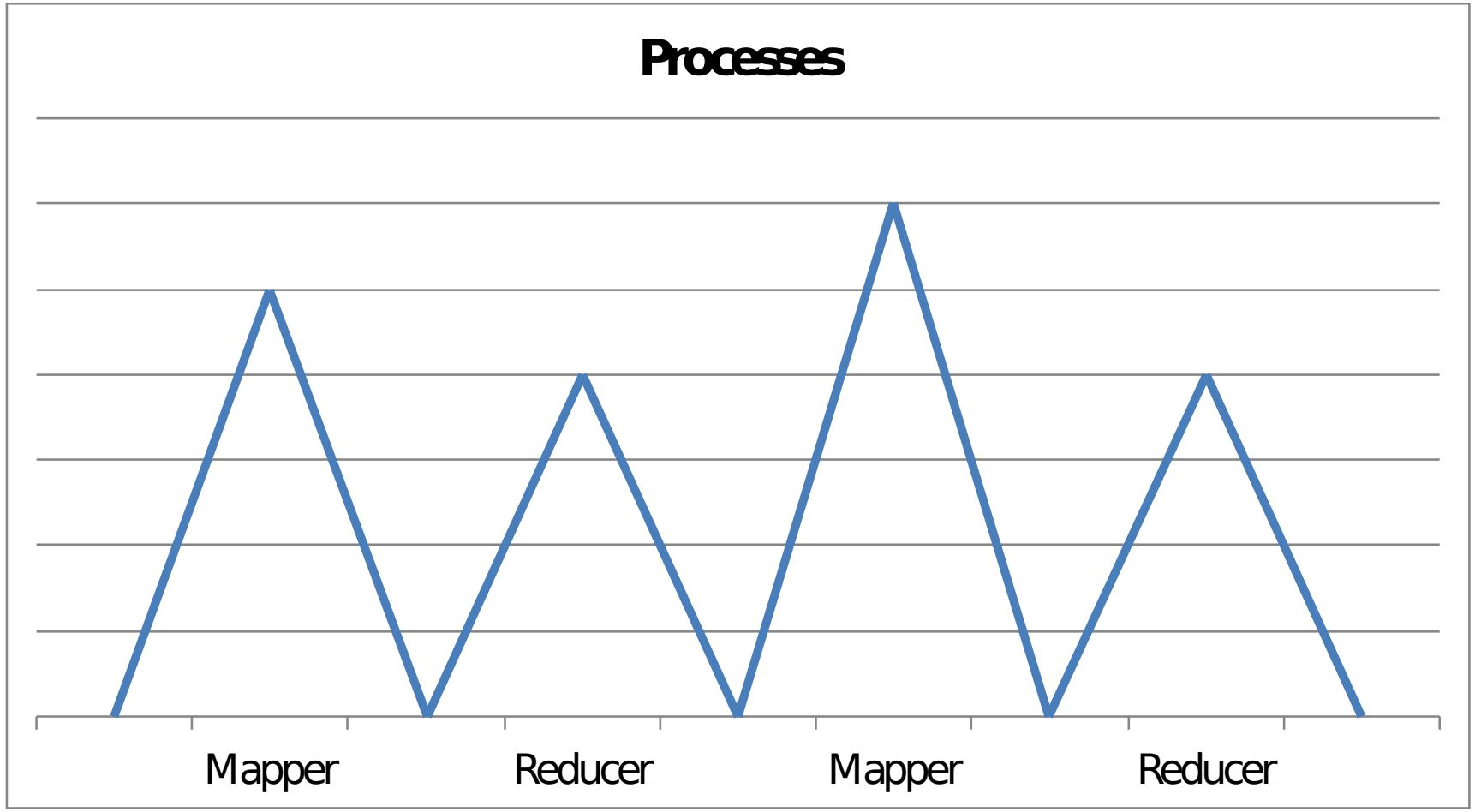
- Select key from table order by key;
  - => Spark “orderBy” transform: range-partition {1,10}, {11,20}, parallel sorting
  - In Mapreduce, used to hash-partition to 1 partition, sort in serial

Improvement: Process Lifecycle

- In MapReduce, each Map/Reduce phase spawns and terminates many processes (Mappers, Reducers)
- In Spark, each “Executor” can be long-lived, runs one or more tasks.
- A set of Spark Executors = Spark “Application”.

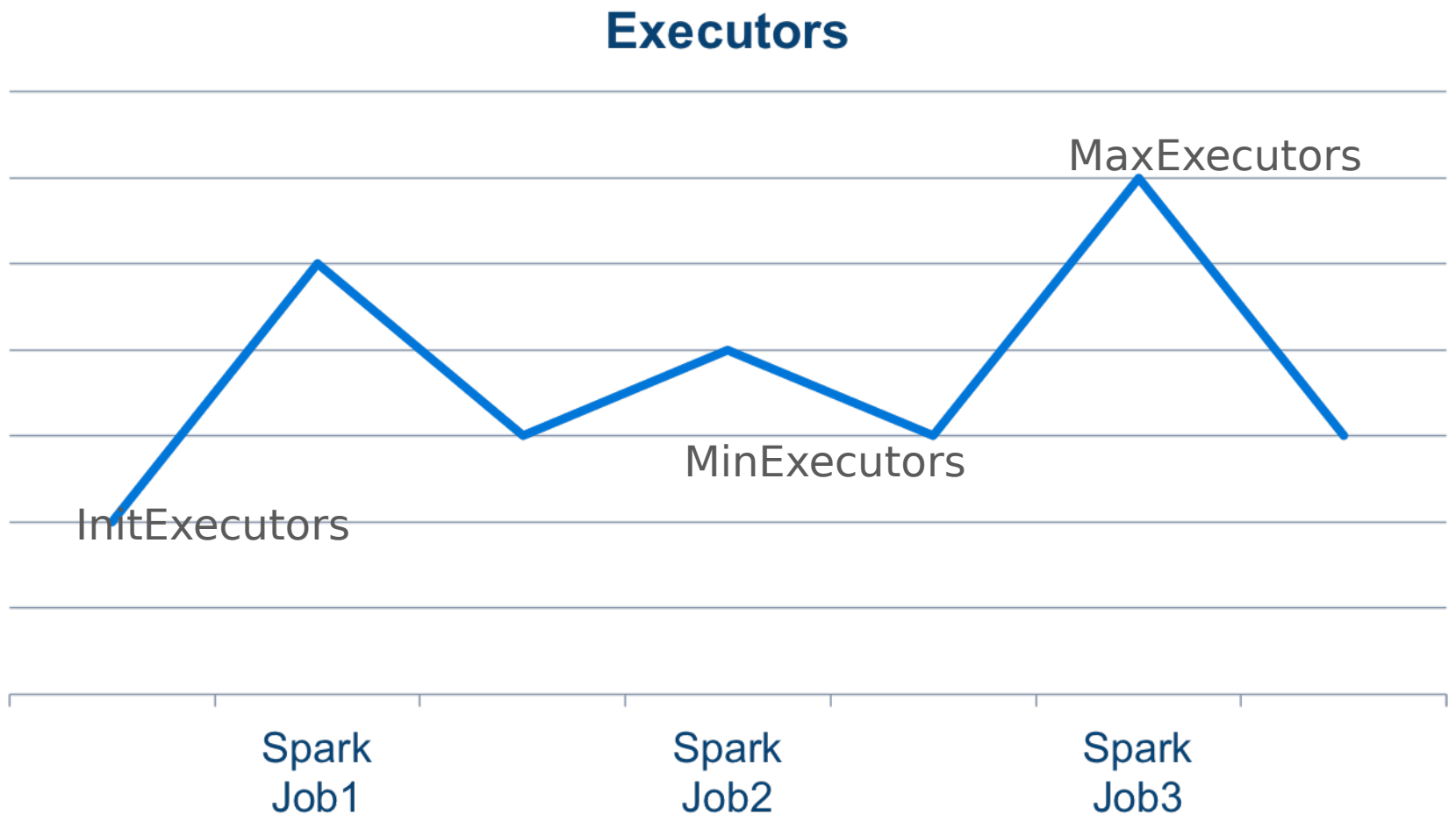
- In Hive on Spark, one Hive user session has open one Spark Application.
  - All queries of that user session re-use Application, can re-use the Executor processes.

Improvement: Process Lifecycle



cloudera

Improvement: Process Lifecycle



- Background: Hive, Spark, Hive on Spark
- Technical Deep Dive
- **User-View**

User View

- Install Hadoop on cluster
  - HDFS
  - YARN (recommended)
- Install Spark (YARN mode recommended)
- Install Hive (will pick up static Spark configs, like spark.master, spark.serializer)
- From Versions: Hive 1.1, Spark 1.3, Hadoop 2.6

- In Hive client, run “Set hive.execution.engine=spark”; //default is MR
- Run query
- The first query will start the Spark application (set of Executors)

User View

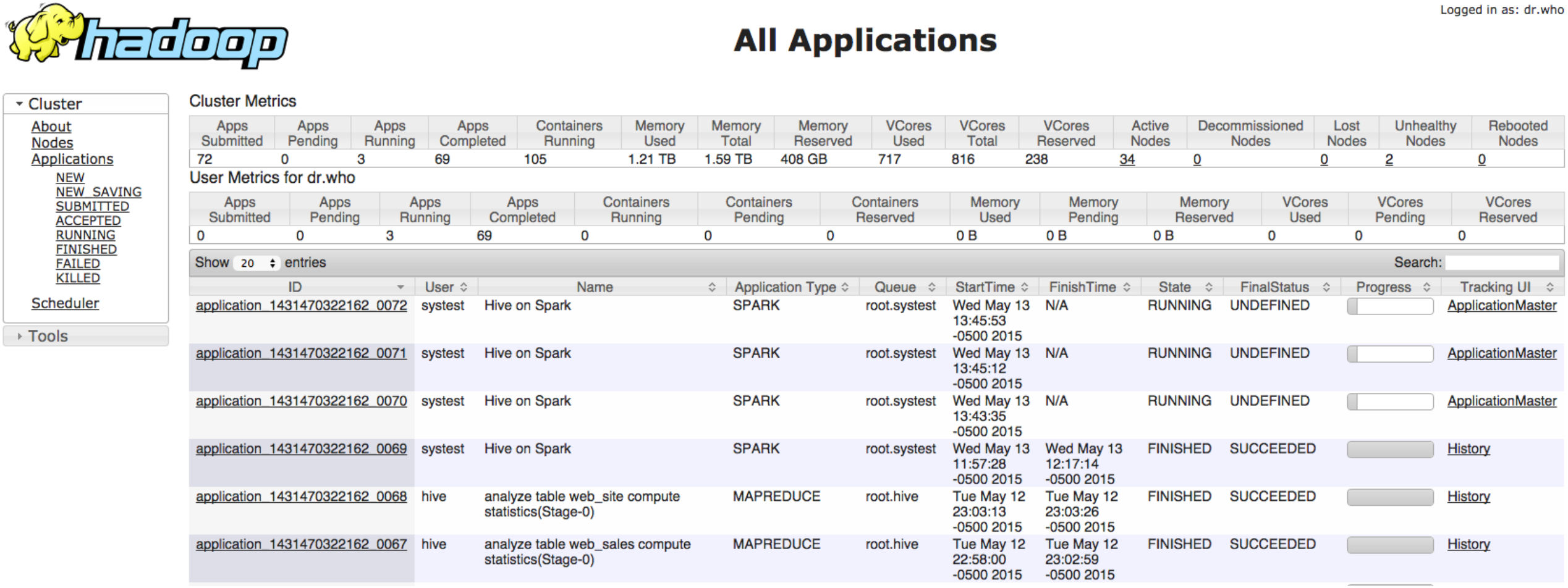
```
0: jdbc:hive2://localhost:10000> select * from store_sales order by ss_item_sk;
INFO  : In order to change the average load for a reducer (in bytes):
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
INFO  : Starting Spark Job = 4158d44f-ec20-4c51-af6a-f8eeb2a6bf3f
INFO  :
Query Hive on Spark job[0] stages:
INFO  : 0
INFO  : 1
INFO  :
Status: Running (Hive on Spark job[0])
INFO  : Job Progress Format
CurrentTime StageId_StageAttemptId: SucceededTasksCount(+RunningTasksCount-FailedTasksCount)/TotalTasksCount [StageCost]
INFO  : 2015-05-13 09:58:03,243 Stage-0_0: 0/5830       Stage-1_0: 0/1
INFO  : 2015-05-13 09:58:04,255 Stage-0_0: 0(+714)/5830 Stage-1_0: 0/1
INFO  : 2015-05-13 09:58:07,285 Stage-0_0: 0(+714)/5830 Stage-1_0: 0/1
INFO  : 2015-05-13 09:58:10,310 Stage-0_0: 0(+714)/5830 Stage-1_0: 0/1
INFO  : 2015-05-13 09:58:12,327 Stage-0_0: 3(+714)/5830 Stage-1_0: 0/1
INFO  : 2015-05-13 09:58:13,336 Stage-0_0: 8(+714)/5830 Stage-1_0: 0/1
INFO  : 2015-05-13 09:58:14,344 Stage-0_0: 9(+714)/5830 Stage-1_0: 0/1
INFO  : 2015-05-13 09:58:15,353 Stage-0_0: 10(+714)/5830        Stage-1_0: 0/1
INFO  : 2015-05-13 09:58:17,369 Stage-0_0: 11(+714)/5830        Stage-1_0: 0/1
INFO  : 2015-05-13 09:58:20,392 Stage-0_0: 11(+714)/5830        Stage-1_0: 0/1
INFO  : 2015-05-13 09:58:23,414 Stage-0_0: 11(+714)/5830        Stage-1_0: 0/1
INFO  : 2015-05-13 09:58:26,435 Stage-0_0: 11(+714)/5830        Stage-1_0: 0/1
INFO  : 2015-05-13 09:58:29,458 Stage-0_0: 11(+714)/5830        Stage-1_0: 0/1
INFO  : 2015-05-13 09:58:32,480 Stage-0_0: 11(+714)/5830        Stage-1_0: 0/1
INFO  : 2015-05-13 09:58:35,503 Stage-0_0: 11(+714)/5830        Stage-1_0: 0/1
INFO  : 2015-05-13 09:58:38,527 Stage-0_0: 11(+714)/5830        Stage-1_0: 0/1
```

Spark job status

cloudera

User View

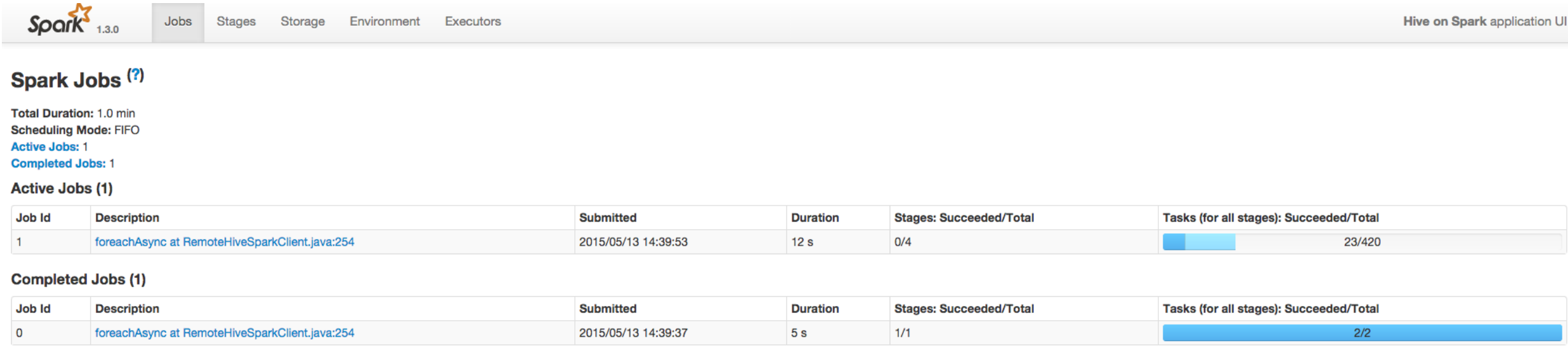
Find your corresponding Spark application in the YARN UI



cloudera

User View

- Click on link to Spark History Server for Corresponding Spark Application progress and information.



Dynamic vs Static Allocation

For a Spark Application:

- Spark dynamic allocation: number of Executor instances variable.
  - spark.executor.dynamicAllocation.enabled=true
  - spark.executor.dynamicAllocation.initialExecutors=1
  - spark.executor.dynamicAllocation.minExecutors=1
  - spark.executor.dynamicAllocation.maxExecutors=10;
- Spark static allocation: number of Executor instances fixed.
  - Spark.executor.instances = 10

User View

- Things to tune: memory of Spark executors
  - spark.executor.cores: number of cores per Spark executor.
  - spark.executor.memory: maximum size of each Spark executor's Java heap memory when Hive is running on Spark.
  - spark.driver.memory: maximum size of each Spark driver's Java heap memory when Hive is running on Spark.

Perf Benchmarks

- 8 physical nodes
- Each node: 32 core, 64 GB
- 10000MB/s network between nodes
- Component Versions
  - Hive: spark-branch (April 2015)
  - Spark: 1.3.0
  - Hadoop: 2.6.0
  - Tez: 0.5.3

Perf Benchmarks

- 320GB and 4TB TPC-DS datasets

- Three engines share the most of the configurations
  - Memory Vectorization enabled
  - CBO enabled
  - hive.auto.convert.join.noconditionaltask.size = 600MB

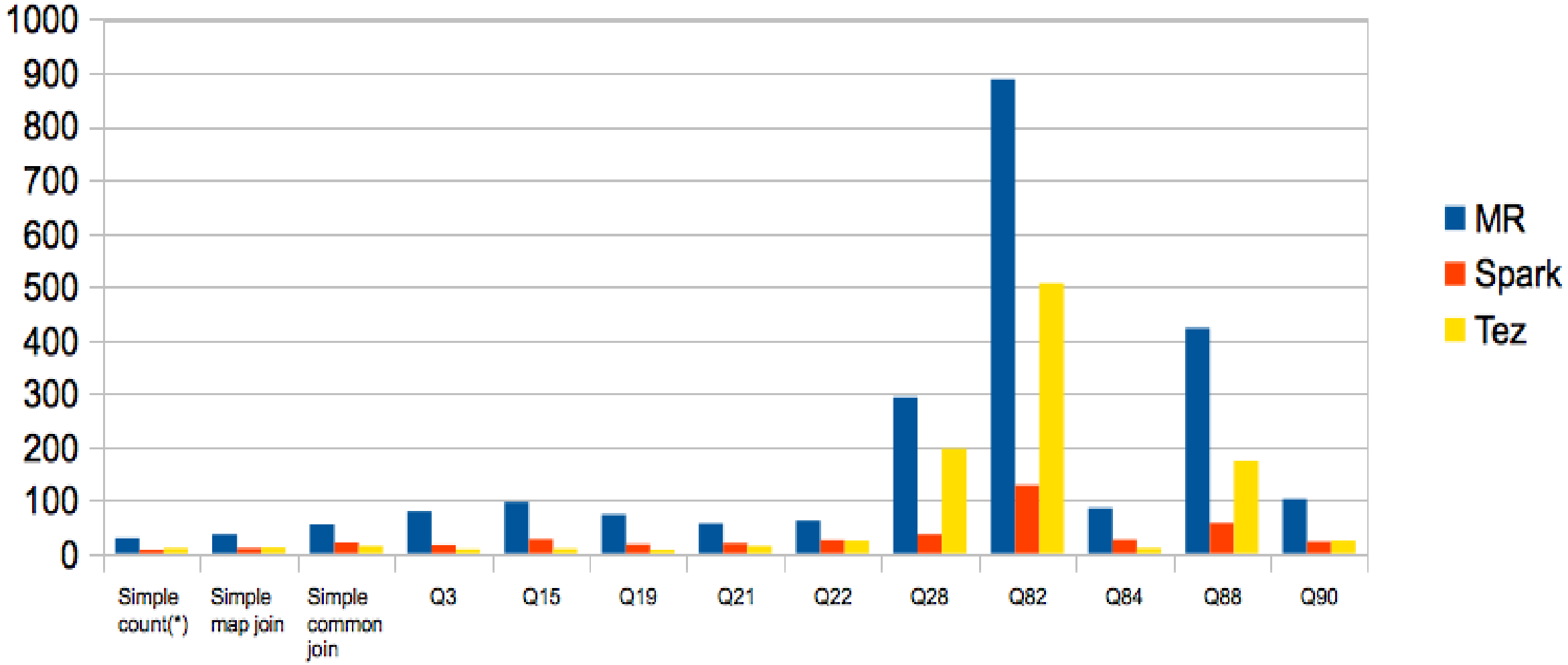
Perf Benchmarks

- Hive on Tez
  - hive.prewarm.numcontainers = 250
  - hive.tez.auto.reducer.parallelism = true
  - hive.tez.dynamic.partition.pruning = true
- Hive on Spark
  - spark.master = yarn-client
  - spark.executor.memory = 5120m
  - spark.yarn.executor.memoryOverhead = 1024
  - spark.executor.cores = 4
  - spark.kryo.referenceTracking = false
  - spark.io.compression.codec = lzf

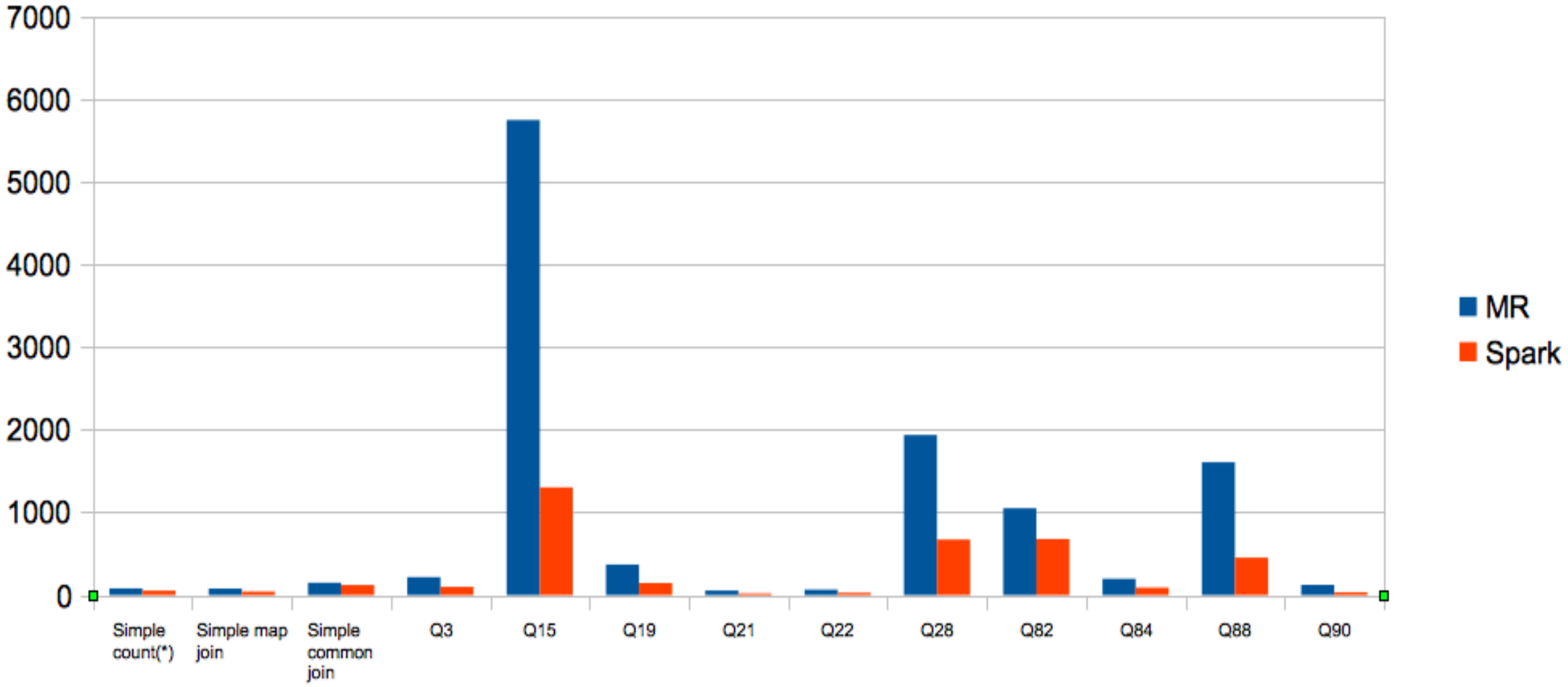
Perf Benchmarks

- Data collection: Run each query twice, first to warm-up, second to measure.

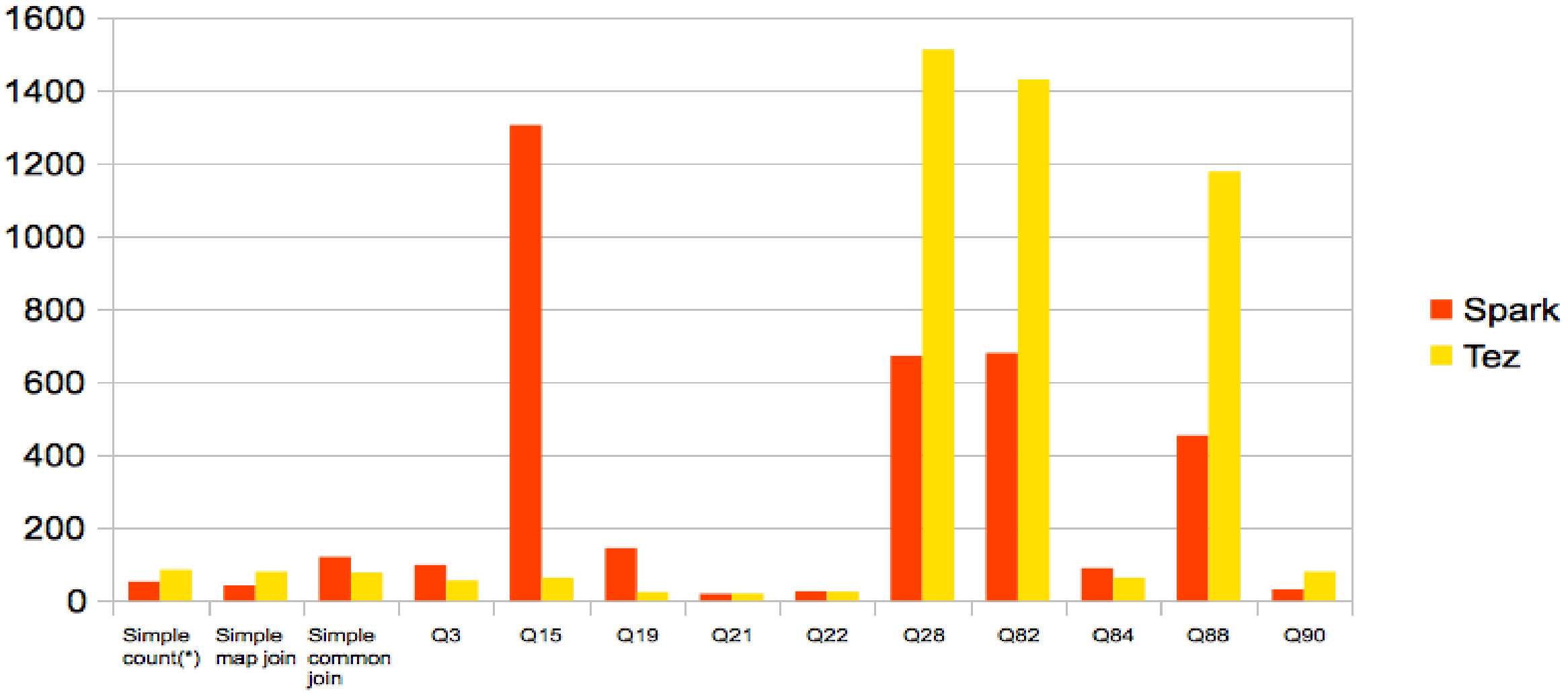
MR vs Spark vs Tez, 320GB



MR vs Spark, 4TB



Spark vs Tez, 4TB

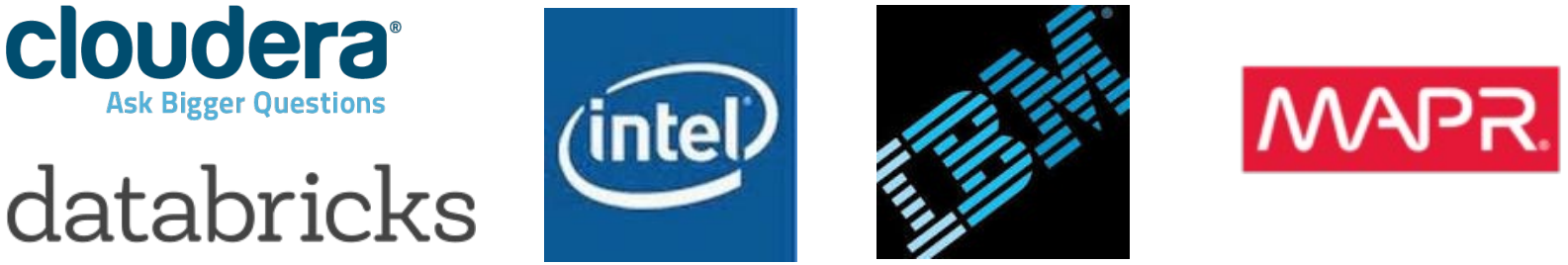


Perf Benchmarks

- Spark is as fastest on many queries
- Dynamic partition pruning makes Spark slower in some queries (Q3, Q15, Q19). These queries benefit from eliminating some partition from bigger-table before a join.
- Spark is slower on certain queries (common join, Q84) than Tez. Spark shuffle-sort improvements in the works in Spark community (Project Tungsten, etc)

Conclusion

- Available in Hive 1.1+, CDH5.4+
- Follow HIVE-7292 for more updates
- Contributors from:



cloudera

Thank you.

SparkSQL and Hive on Spark

- SparkSQL is similar to Shark (discontinued)
- Forked a version from Hive, thus tied with a specific version
- Executing queries using Spark's transformations and actions, instead of Hive operators.
  - All SQL syntaxes, functionality implemented from scratch.
- Relatively new
- Suitable for Spark users occasionally needing to execute SQL

Impala?

- Tuned for extreme performance/ low latency
- Purpose-built for interactive BI and SQL analytics
- Best for high concurrency workloads and small result sets