d

Hive on Spark: What It Means To You

Xuefu Zhang
Cloudera
Apache Hive PMC

Short History

- Incepted in Summer 2014
- Tracked in HIVE-7292
- Released in Apache Hive 1.1 in March 2015
- First CDH Beta in 5.3
- Suppor provided for selected customers in 5.4

© Cloudera, Inc. All rights reserved.

Current Status

- Feature development is completed
- Extensive scale and stress testing
- Performance testing
- A lot of optimization work ongoing

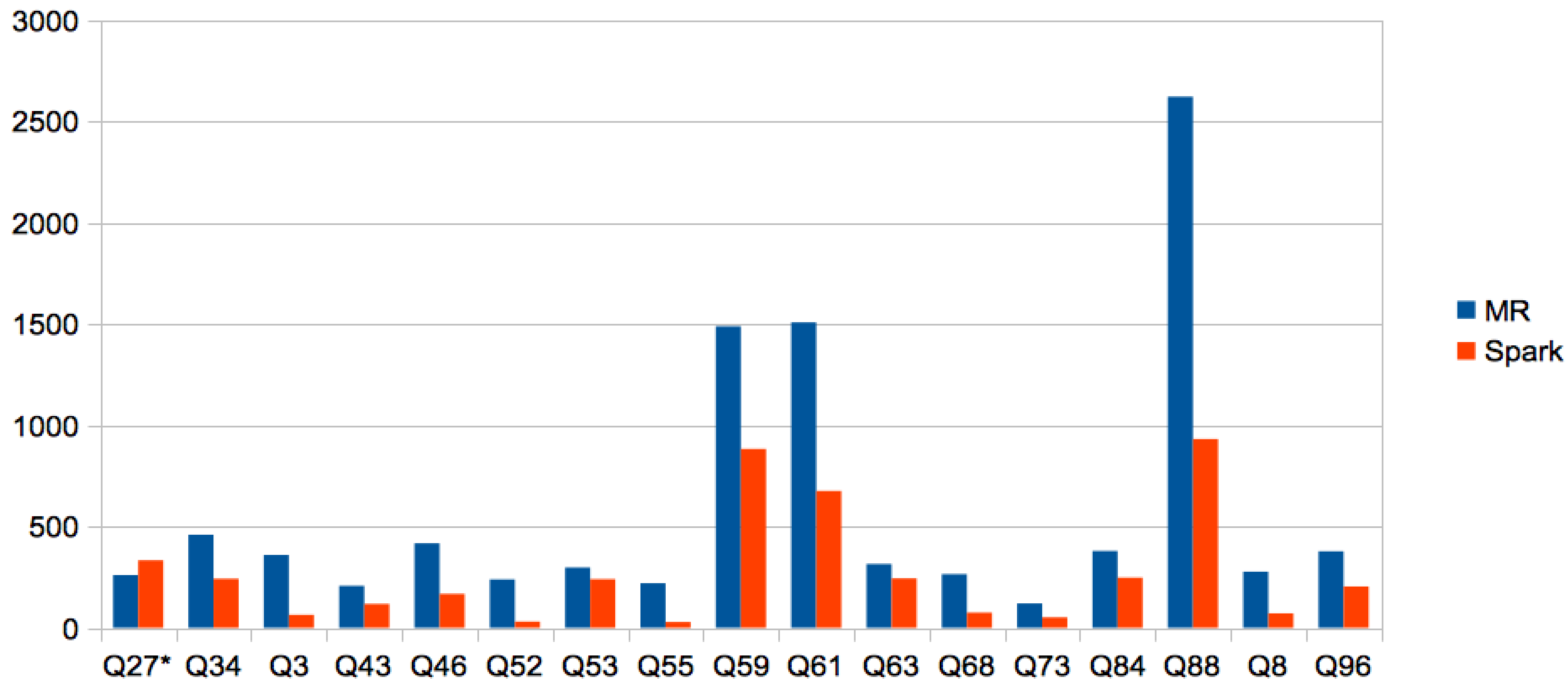
© Cloudera, Inc. All rights reserved.

Performance Overview

- Compared to MapReduce in CDH5.4
- 40 node cluster
- 10T dataset

© Cloudera, Inc. All rights reserved.

Performance Overview (cont'd)



© Cloudera, Inc. All rights reserved.

Performance Overview (cont'd)

- On average, a few times faster than MapReduce
- Result comes w/o any fine tuning
- Q27 is slower due to map join table size threshold. Adjusting size converts map join for Spark as well, 72s vs 261s

© Cloudera, Inc. All rights reserved.

Benefits

- Native in Hive
- All queries work as before
- Share common features in Hive
- Minimum configurations
- Better performance

© Cloudera, Inc. All rights reserved.

Configuration Guide

- Assume a 40 node cluster for YARN
- A few gateway/service node extra
- Each node has 32 cores and 120G memory
- No other service other than YARN running on the 40 nodes

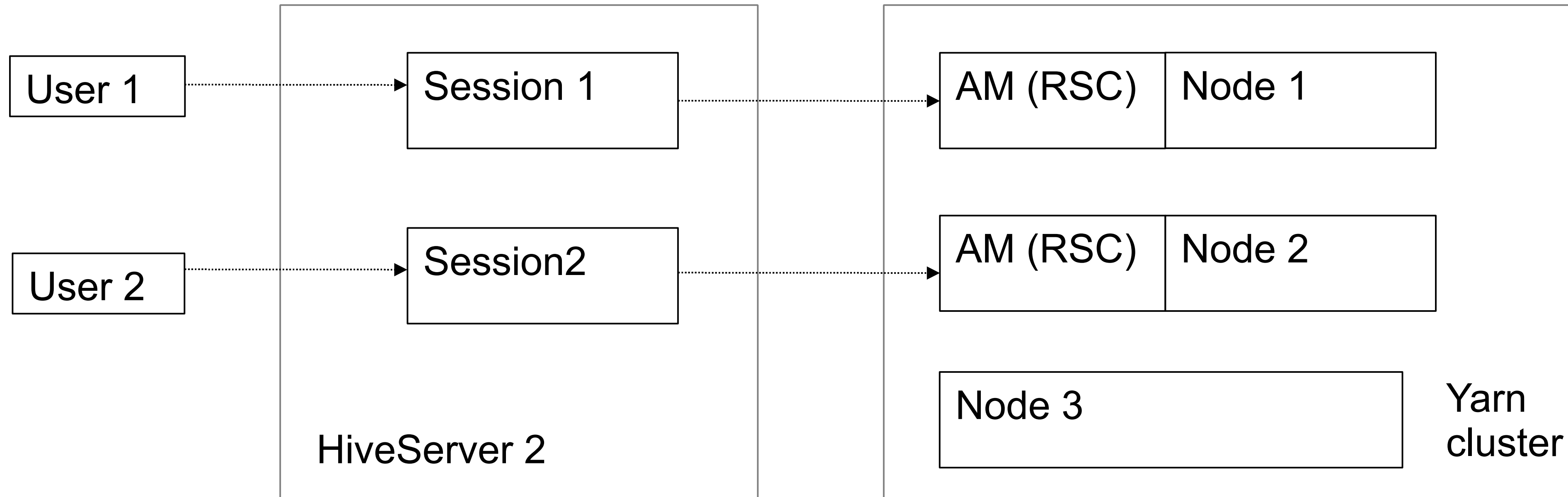
© Cloudera, Inc. All rights reserved.

Basic Configurations

- `hive.execution.engine=spark`
- `Spark.master=yarn-cluster`
- Other modes work, though `yarn-cluster` is highly recommended/supported

© Cloudera, Inc. All rights reserved.

System Deployment Architecture



© Cloudera, Inc. All rights reserved.

Yarn Configurations

- Two important properties
  - `yarn.nodemanager.resource.cpu-vcores`
  - `yarn.nodemanager.resource.memory-mb`
- They represents the resources (cpu, memory) allocated for YARN
- Largely determined by the capacity of the node

© Cloudera, Inc. All rights reserved.

Yarn Configurations (cont'd)

- Leave 4 cores and 20G memory for NM and OS. Thus,
  - `yarn.nodemanager.resource.cpu-vcores=28`
  - `yarn.nodemanager.resource.memory-mb=100G`
- This is for illustration purpose
- Your mileage may vary
- Refer to YARN configuration guide

© Cloudera, Inc. All rights reserved.

Executor Cores

- Determine parallelism of task execution in a single JVM
- Best if `5`, `6`, or `7`.
- Goal: to minimize unused cores in a NodeManager
- In our example
  - `spark.executor.cores=7`
  - If `6`, waste `4` cores
  - If `5`, waste `3` cores

© Cloudera, Inc. All rights reserved.

Spark Configurations

- Defines how Spark utilizes YARN resources (core, memory)
  - `spark.executor.cores`
  - `spark.executor.memory` and `spark.yarn.executor.memoryOverhead`
  - `spark.driver.memory` and `spark.yarn.driver.memoryOverhead`
- Executor core and memory allocation directly impacts performance

© Cloudera, Inc. All rights reserved.

Executor Memory

- Total memory that can be utilized by an executor
- Divide YARN memory among the executors
- Allocate 15-20% total memory for overhead
- Remaining goes to executor memory
- In our example,
  - `4` executors per node (`28/7`)
  - `spark.executor.memory=20G`
  - `spark.yarn.executor.memoryOverhead=5G`

© Cloudera, Inc. All rights reserved.

Driver Memory

- Total memory that can be used by the driver (Spark remote context)
- Also determined based on YARN memory (X)
  - 12GB when X is greater than 50GB
  - 4GB when X is between 50GB and 12GB
  - 1GB when X is between 12GB and 1G
  - 256MB when X is less than 1GB
- Allocate 10-15% total memory for overhead
- Remaining goes to driver memory

© Cloudera, Inc. All rights reserved.

Driver Memory (cont'd)

- In our example
  - `spark.driver.memory=10.5G`
  - `spark.yarn.driver.memoryOverhead=1.5G`
- This just serves as a general guideline
- No impact on performance in general

© Cloudera, Inc. All rights reserved.

Executor Allocation

- Total number of executors is the resource for Hive queries
  - In the example, we have `4x40=160`
  - Each executor can run up to `7` tasks
  - Thus, there can be `160x7=1120` tasks running concurrently
- Executor resource is usually shared in a production deployment
- When allocated, an executor belongs to a specific Hive user session
- How to share exectors: static vs dynamic

© Cloudera, Inc. All rights reserved.

Static Allocation

- Fixed number of executors for a user session
  - `spark.executor.instances`
- Seems fair but inefficient usage
- Heavy users vs light users
- Big queries vs small queries
- Will not be released until user session is gone

© Cloudera, Inc. All rights reserved.

Dynamic Allocation

- Allocates executors dynamically based on load
  - `spark.dynamicAllocation.enabled=true`
  - `spark.dynamicAllocation.initialExecutors`
  - `spark.dynamicAllocation.minExecutors`
  - `spark.dynamicAllocation.maxExecutors`
- Allocates more when needed
- Gives up executors when idle

© Cloudera, Inc. All rights reserved.

Performance Tuning

- Directly related to but not necessarily proportional to the number of executors
- Maximum number of executors usually gives optimal or close to optimal performance
- Strongly suggested when you do benchmarking
- Half of the maximum executors usually gives a reasonable good performance

© Cloudera, Inc. All rights reserved.

Performance Tuning (cont'd)

- Share most of existing performance related configurations
- One important exception
  - hive.auto.convert.join.noconditionaltask.size
- MR uses file size
- Spark uses raw data size, closer to the amount of memory needed
- Compression technology can make them dramatically diverge
- You might need to increase the value to be at the same level as MR

© Cloudera, Inc. All rights reserved.

Short-Lived Sessions

- Noticeable delay before job launch
  - Executor starts up and initializes
- Running the same query for the second time usually gives much better performance
  - No delay in launching executors
  - Higher parallelism in reduce stages
- This gives a disadvantage for short-lived sessions
  - Hive action lanuched by Oozie

© Cloudera, Inc. All rights reserved.

Short-Lived Sessions (cont'd)

- Prewarm YARN container (spark executor)
  - Trade slightly longer delay (a few seconds) for better parallelism
- Tow configurations
  - `hive.prewarm.enabled`
  - `hive.prewarm.numcontainers`
- `hive.prewarm.numcontainers` should be based on executor allocation
- Wrong configuration may cause a maximum of 30s delay

© Cloudera, Inc. All rights reserved.

General Performance Tuning

- Most of performance tuning params also applies to Spark
- For example
  - `hive.stats.fetch.column.stats=true`
  - `hive.optimize.index.filter=true`
- There are much more

© Cloudera, Inc. All rights reserved.

General Performance Tuning (cont'd)

- Consider tuning performance in the following categories
  - Enable vectorization
  - Enable CBO
  - Reducer de-duplication
  - Merge small files (one specific for Spark)
  - Mapjoin auto conversion
  - SMJ

© Cloudera, Inc. All rights reserved.

General Performance Tuning (cont'd)

- Much more
  - Map-Side aggregation
  - Collect and use as much statistics as possible
  - Use index filter
  - Size per reducer (less impact on Spark)

© Cloudera, Inc. All rights reserved.

Trouble-Shooting

- Console output
- HiveServer2 log
- YARN application log
- YARN container log
- Spark driver webui and history server
  - Especially useful to see task statistics
  - Find slow nodes or long-trailing tasks

© Cloudera, Inc. All rights reserved.

Summary

- Bearing solid foundations in design and architecture
- Extensively tested, production ready
- Great initial Performance
- Benefiting both Hive users and Spark users
- We welcome your feedback and contributions

© Cloudera, Inc. All rights reserved.

References

- https://issues.apache.org/jira/browse/HIVE-7292
- CDH5.4 download: http://www.cloudera.com/content/cloudera/en/downloads.html
- Hive on Spark “Getting Started” http://www.cloudera.com/content/cloudera/en/documentation/hive-spark/latest
- Community support via user@hive.apache.org
- http://community.cloudera.com/

© Cloudera, Inc. All rights reserved.

SparkSQL?

- SparkSQL is similar to Shark (discontinued)
- Uses Hive's metastore and so is tied to a specific version
- Executing queries using Spark's transformations and actions
- Support a subset of Hive's syntax and functionality
- Immature
- Suitable for Spark developers (Scala/Java) occasionally executing SQL

© Cloudera, Inc. All rights reserved.

Demo, Q&A

© Cloudera, Inc. All rights reserved.