Hortonworks Data Platform

Performance Tuning Guide

(Jul 18, 2014)

docs.hortonworks.com

Hortonworks Data Platform: Performance Tuning Guide

The Hortonworks Data Platform, powered by Apache Hadoop, is a massively scalable and 100% open source platform for storing, processing and analyzing large volumes of data. It is designed to deal with data from many sources and formats in a very quick, easy and cost-effective manner. The Hortonworks Data Platform consists of the essential set of Apache Hadoop projects including MapReduce, Hadoop Distributed File System (HDFS), HCatalog, Pig, Hive, HBase, Zookeeper and Ambari. Hortonworks is the major contributor of code and patches to many of these projects. These projects have been integrated and tested as part of the Hortonworks Data Platform release process and installation and configuration tools have also been included.

Unlike other providers of platforms built using Apache Hadoop, Hortonworks contributes 100% of our code back to the Apache Software Foundation. The Hortonworks Data Platform is Apache-licensed and completely open source. We sell only expert technical support, training and partner-enablement services. All of our technology is, and will remain free and open source.

Please visit the Hortonworks Data Platform page for more information on Hortonworks technology. For more information on Hortonworks services, please visit either the Support or Training page. Feel free to Contact Us directly to discuss your specific needs.

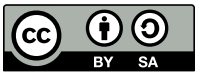


Table of Contents

List of Tables

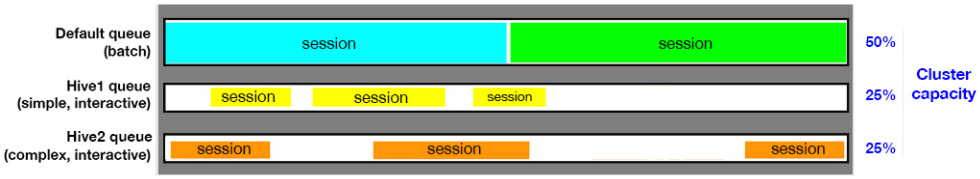
1. Tuning for Interactive and Batch Queries

Queues are the primary method used to manage multiple workloads. Queues can provide workload isolation and thus guarantee available capacity for different workloads. This can help different types of workloads meet Service Level Agreements (SLAs).

Within each queue, you can allow one or more sessions to live simultaneously. Sessions cooperatively share the resources of the queue.

For example, a queue that is assigned 10% of cluster resources can get these allocated anywhere in the cluster depending on the query and data placement. Where the resources are allocated may change as more queries are run.

The following figure shows a configuration in which 50% of the cluster capacity is assigned to a "default" queue for batch jobs, along with two queues for interactive Hive queries, with each Hive queue assigned 25% of cluster resources. Two sessions are used in the batch queue, and three sessions are used in each Hive queue.



1.1. Tuning for Interactive Hive Queries

The following general guidelines are recommended for interactive Hive queries. The YARN, Tez, and HiveServer2 configuration settings used to implement these guidelines are discussed in more detail in subsequent sections of this document.

- **Limit the number of queues** – Because Capacity Scheduler queues allocate more-or-less fixed percentages of cluster capacity, Hortonworks recommends configuring clusters with a few small-capacity queues for interactive queries. Queue capacity can be configured to be flexible, and a good use case for that can be to allow batch processes to take over the entire cluster at night when other workloads are not running.

- **Allocate queues based on query duration** – For example, say you have two types of commonly used queries. One type of query takes about 5 seconds to run, and the other type takes about 45 seconds to run. If both of these types were assigned to the same queue, the shorter queries might have to wait for the longer queries. In this case it would make sense to assign the shorter queries to one queue, and the longer queries to another queue.

- **Reuse containers to increase performance** – Enabling Tez container reuse improves performance by avoiding the memory overhead of reallocating container resources for every task.

- **Configure Query Vectorization** – Query vectorization helps reduce execution time for interactive queries.
- **Use sessions to allocate resources within individual queues** – rather than increasing the number of queues.

1.1.1. Create and Configure YARN Capacity Scheduler Queues

Capacity Scheduler queues can be used to allocate cluster resources among users and groups. These settings can be accessed from **Ambari > YARN > Configs > Scheduler** or in `capacity-scheduler.xml`. YARN must be restarted in order for queues to take effect.

To demonstrate how to set up Capacity Scheduler queues, let's use the following simple configuration that might be used to separate short and long-running queries into two separate queues.

- hive1 – this queue will be used for short-duration queries, and will be assigned 50% of cluster resources.
- hive2 – this queue will be used for longer-duration queries, and will be assigned 50% of cluster resources.

The following `capacity-scheduler.xml` settings in would be used to implement this configuration:

```
yarn.scheduler.capacity.root.queues=hive1,hive2
yarn.scheduler.capacity.root.hive1.capacity=50
yarn.scheduler.capacity.root.hive2.capacity=50
```

Let's also set limits on usage for these queues and their users:

```
yarn.scheduler.capacity.root.hive1.maximum-capacity=50
yarn.scheduler.capacity.root.hive2.maximum-capacity=50
yarn.scheduler.capacity.root.hive1.user-limit=1
yarn.scheduler.capacity.root.hive2.user-limit=1
```

The value of "50" for `maximum-capacity` means that queue users are restricted to 50% of the queue capacity (hard limit). If the `maximum-capacity` were more than 50%, the queue could use more than its capacity when there are other idle resources in the cluster. However, any one user can still only use up to the configured queue capacity. The default value of "1" for `user-limit` means that any single user in the queue can at maximum only occupy 1x the queue's configured capacity. These settings prevent users in any one queue from monopolizing resources across all queues in a cluster.

The preceding example represents a very basic introduction to queues. For more detailed information on allocating cluster resources using Capacity Scheduler queues, see Capacity Scheduler.

1.1.2. Configure Tez Container Reuse

Tez settings can be accessed from **Ambari >Tez > Configs > Advanced** or in `tez-site.xml`.

Enabling Tez container reuse improves performance by avoiding the memory overhead of reallocating container resources for every task. This can be achieved by having the queue retain resources for a specified amount of time, so that subsequent queries run faster.

For good performance with smaller interactive queries on a busy cluster, you might retain resources for 5 minutes. On a less busy cluster, or if consistent timing is very important, you might hold on to resources for 30 minutes.

The following settings can be used to configure Tez to enable container reuse.

- **Tez Application Master Waiting Period** (in seconds) – Specifies the amount of time in seconds that the Tez Application Master (AM) waits for a DAG to be submitted before shutting down. For example, to set the waiting period to 15 minutes (15 minutes x 60 seconds per minute = 900 seconds):

  ```
  tez.session.am.dag.submit.timeout.secs=900
  ```

- **Enable Tez Container Reuse** – This configuration parameter determines whether Tez will reuse the same container to run multiple queries. Enabling this parameter improves performance by avoiding the memory overhead of reallocating container resources for every query.

  ```
  tez.am.container.reuse.enabled=true
  ```

- **Tez Container Holding Period** (in milliseconds) – Specifies the amount of time in milliseconds that a Tez session will retain its containers. For example, to set the holding period to 15 minutes (15 minutes x 60 seconds per minute x 1000 milliseconds per second = 900000 milliseconds):

  ```
  tez.am.container.session.delay-allocation-millis=900000
  ```

  A holding period of a few seconds is preferable when multiple sessions are sharing a queue. However, a short holding period negatively impacts query latency.

For more information on these and other Tez configuration settings, see Configure Tez.

Note

Do not use the `tez.queue.name` configuration parameter because it sets all Tez jobs to run on one particular queue.

Confirming Container Reuse

To confirm container reuse, run a query, then reload the UI. You should see some number of containers being used. The second or third time you run the query, no allocation of containers should be needed, and the query should run more quickly.

1.1.3. Configure HiveServer2 Settings

HiveServer2 is used for remote concurrent access to Hive. HiveServer2 settings can be accessed from **Ambari >Tez > Configs > Advanced** or in `hive-site.xml`. You must restart HiveServer2 in order for updated settings to take effect.

- **Hive Execution Engine** – Set this to "tez" to execute Hive queries using Tez:

```
hive.execution.engine=tez
```

- **Enable Default Sessions** – Uses a default session for jobs using HiveServer2 even if they don't use Tez. Set this to "true".

```
hive.server2.tez.initialize.default.sessions=true
```

- **Specify the HiveServer2 Queues** – A comma-separated list of queues. For example, to specify queues "hive1" and "hive2":

```
hive.server2.tez.default.queues=hive1,hive2
```

- **Set the Number of Sessions in each Queue** – Sets the number of sessions for each queue named in hive.server2.tez.default.queues.

```
hive.server2.tez.sessions.per.default.queue=1
```

- **Set enable.doAs to False** – We want enable.doAs to be false since this uses the Hive user identity rather than the individual user identities for YARN. This helps with security and reuse.

```
hive.server2.enable.doAs=false
```

- **Configure Query Vectorization** – The following two settings help reduce execution time for interactive queries on small datasets, but are also okay for large datasets.

```
hive.vectorized.groupby.maxentries=10240
hive.vectorized.groupby.flush.percent=0.1
```

For more information on these and other HiveServer2-on-Tez configuration settings, see Configure Hive and HiveServer2 for Tez.

1.1.4. Adjusting Settings for Increasing Number of Concurrent Users

As the number of concurrent users increases, you should generally keep the number of queues to a minimum and instead increase the number of sessions in each queue. For example, for 5-10 concurrent users, 2-5 queues with 1-2 sessions each might be adequate. To set 3 queues with 2 sessions for each queue:

```
hive.server2.tez.default.queues=hive1,hive2,hive3
hive.server2.tez.sessions.per.default.queue=2
```

If the number of concurrent users increases to 15, you may achieve better performance using 5 queues with 3 sessions per queue:

```
hive.server2.tez.default.queues=hive1,hive2,hive3,hive4,hive5
hive.server2.tez.sessions.per.default.queue=3
```

The following table provides general guidelines for the number of queues and sessions for increasing numbers of concurrent users.

Table 1.1. Queues and Sessions for Increasing Numbers of Concurrent Users

Number of Users	Number of Concurrent Users	Number of Queues	Number of Sessions per Que
50	5	2-5	1-2

Number of Users	Number of Concurrent Users	Number of Queues	Number of Sessions per Que
100	10	5	2
150	15	5	3
200	20	5	4

1.2. Tuning for a Mix of Interactive and Batch Hive Queries

In general, adjustments for interactive queries will not adversely affect batch queries, so both types of queries can usually run well together on the same cluster. You can use Capacity Scheduler queues to divide cluster resources between batch and interactive queries. For example, you might set up a configuration that allocates 50% of the cluster capacity to a "default" queue for batch jobs, and two queues for interactive Hive queries, with each assigned 25% of cluster resources:

```
yarn.scheduler.capacity.root.queues=default,hive1,hive2
yarn.scheduler.capacity.root.default.capacity=50
yarn.scheduler.capacity.root.hive1.capacity=25
yarn.scheduler.capacity.root.hive2.capacity=25
```

The following settings would enable the capacity of the batch queue to expand to 100% when no one else is using the cluster (at night, for example). The maximum-capacity of the default (batch) queue is set to 100%, and the user-limit-factor is increased to 2 to enable the queue users to occupy twice the queue's configured capacity of 50%.

```
yarn.scheduler.capacity.root.default.maximum-capacity=100
yarn.scheduler.capacity.root.default.user-limit-factor=2
```