Cloudera Runtime 7.2.18

Managing Apache Impala

Date published: 2020-11-30
Date modified: 2024-03-25

https://docs.cloudera.com/

Legal Notice

© Cloudera Inc. 2026. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

READ Support for FULL ACID ORC Tables

FULL ACID v2 transactional tables are readable in Impala without modifying any configurations. You must have Cloudera Runtime 7.2.2 or higher and have connection to Hive Metastore server in order to READ from FULL ACID tables.

There are two types of transactional tables available with Hive ACID.

- INSERT-ONLY
- FULL ACID

Until this release, Impala in Cloudera supported INSERT-ONLY transactional tables allowing both READ and WRITE operations. The latest version of Impala in Cloudera now also supports READ of FULL ACID ORC tables.

By default tables created in Impala are INSERT-ONLY managed tables whereas the default tables in Hive are managed tables that are FULL-ACID and INSERT-ONLY.

Limitations

- Impala cannot CREATE or WRITE to FULL ACID transactional tables yet. You can CREATE and WRITE FULL ACID transactional tables with transaction scope at the row level via HIVE and use Impala to READ these tables.
- Impala does not support ACID v1.

Concepts Used in FULL ACID v2 Tables

Before beginning to use FULL ACID v2 tables you must be aware of these new concepts like transactions, WriteIds, rowIDs, delta delete directories, locks, etc. that are added to FULL ACID tables to achieve ACID semantics.

Write IDs

For every transaction, both read and write, Hive will assign a globally unique ID. For transactional writes like INSERT and DELETE, it will also assign a table-wise unique ID, a write ID. The write ID range will be encoded in the delta and delete directory names. Results of a DML transactional query are allocated to a location under partition/table. This location is derived by Write ID allocated to the transaction. This provides Isolation of DML queries and such queries can run in parallel without interfering with each other.

New Sub-directories

New data files resulting from a DML query are written to a unique location derived from WriteId of the transaction. You can find the results of an INSERT query in delta directories under partition/table location. Depending on the operation type there can be two types of delta directories:

- Delta Directory: This type is created for the results of INSERT statements and is named delta_<writeId>_<writeId> under partition/table location.
- Delete Delta Directory: This delta directory is created for results of DELETE statements and is named delete_delta_<writeId>_<writeId> under partition/table location.

UPDATE operations create both delete and delta directories.

Row IDs

rowId is the auto-generated unique ID within the transaction and bucket. This is added to each row to identify each row in a table. RowID is used during a DELETE operation. When a record is deleted from a table, the rowId of the deleted row will be written to the delete_delta directory. So for all subsequent READ operations all rows will be read except these rows.

Schematic differences between INSERT-ONLY and FULL ACID tables

INSERT-ONLY tables do not have a special schema. They store the data just like plain original files from the non-ACID world. However, their files are organized differently. For every INSERT statement the created files are put into a transactional directory which has transactional information in its name.

Full ACID tables do have a special schema. They have row identifiers to support row-level DELETEs. So a row in Full ACID format looks like this:

```
{
   "operation": 0,
   "originalTransaction": 1,
   "bucket": 536870912,
   "rowId": 0,
   "currentTransaction": 1,
   "row": {"i": 1}
   }
```

- The green columns are the hidden/system ACID columns.
- Field “row” holds the user data.
- operation 0 means INSERT, 1 UPDATE, and 2 DELETE. UPDATE will not appear because of the split-update technique (INSERT + DELETE).
- originalTransaction is the write ID of the INSERT operation that created this row.
- bucket is a 32-bit integer defined by BucketCodec class.
- rowId is the auto-generated unique ID within the transaction and bucket.
- currentTransaction is the current write ID. For INSERT, it is the same as currentTransaction. For DELETE, it is the write ID when this record is first created.
- row contains the actual data. For DELETE, row will be null.

Key Differences between INSERT-ONLY and FULL ACID Tables

Before beginning to use FULL ACID v2 tables you must be aware of the key differences between the INSERT-ONLY and FULL-ACID tables.

This table highlights some of the differences between the INSERT-ONLY and FULL ACID tables.

	INSERT-ONLY	FULL ACID
Schema	There is no special data schema. They store the data just like plain original files from the non-ACID world.	Data is in special format, i.e. there are synthetic columns with transactional information in addition to actual data.
Transactional information	Transactional information is encoded in directory names.	Full ACID tables also use the same directory structure as INSERT-only tables. Transactional information is encoded in the directory names. Directory name and filename are the source of transactional information.
Table properties	'transactional'='true', 'transactional_properties'='insert_only'	'transactional'='true'
Supported operations	INSERT-ONLY tables only support insertion of data. UPDATES and DELETES are not supported. These tables also provide CREATE TABLE, DROP TABLE, TRUNCATE, INSERT, SELECT operations.	FULL ACID ORC tables can be READ using IMPALA. These tables also provide UPDATE and DELETE operations at the row level using HIVE. This is achieved using transactions like Insert-Only Tables along with changes in ORC Reader to support deletes.
WRITE operation	WRITE operations are atomic and the results of the insert operation are not visible to other query operations until the operation is committed.	WRITE operations are atomic - The operation either succeeds completely or fails; it does not result in partial data.

	INSERT-ONLY	FULL ACID
INSERT operation	For every INSERT statement the created files are added to a transactional directory which has transactional information in its name.	INSERT operation is done through HIVE and this statement is executed in a single transaction. This operation creates a delta directory containing information about this transaction and its data.
DELETE operation	N/A	DELETE operation is done through HIVE and this event creates a special “delete delta” directory.
UPDATE operation	N/A	UPDATE operation is done through HIVE. This operation is split into an INSERT and DELETE operation. This operation creates a delta dir followed by a delete dir.
READ operation	READ operations always read a consistent snapshot of the data.	READ operations always read a consistent snapshot of the data.
Supported file format	Supports any file formats.	Supports only ORC.
Compactions	Minor and major compactions are supported.	Minor compactions can be created, which means several delta and delete directories can be compacted into one delta and delete directory. Major compactions are also supported.

Note: Currently, ALTER TABLE statement is not supported on both insert-only and full acid transactional tables.

File structure of FULL ACID transational table

Hive 3 achieves atomicity and isolation of operations on transactional tables by using techniques in write, read, insert, create, delete, and update operations that involve delta files, which can provide query status information and help you troubleshoot query problems.

Compaction of Data in FULL ACID Transactional Table

As administrator, you need to manage compaction of delta files that accumulate during data ingestion. Compaction is a process that performs critical cleanup of files.

Hive creates a set of delta files for each transaction that alters a table or partition and stores them in a separate delta directory. When the number of delta and delete directories in the table grow, the read performance will be impacted, since reading is a process of merging the results of valid transactions. To avoid any compromise on the read performance, occasionally Hive performs compaction, namely minor and major. This process merges these directories while preserving the transaction information.

To initiate automatic compaction, you must enable it using Cloudera Manager. For more information on managing the compaction process, see the link provided under Related Information.

Related Information

Data Compaction

Managing Resources in Impala

Impala includes the features that balance and maximize resources to improve query performance and scalability of your Cloudera cluster.

A typical deployment uses the following resource management features:

- Static service pools

  Use the static service pools to allocate dedicated resources for Impala to manage and prioritize workloads on clusters.
- Admission control

Within the constraints of the static service pool, you can further subdivide Impala's resources using dynamic resource pools and admission control.

Related Information

Creating Static Pools

Dynamic Resource Pool Settings

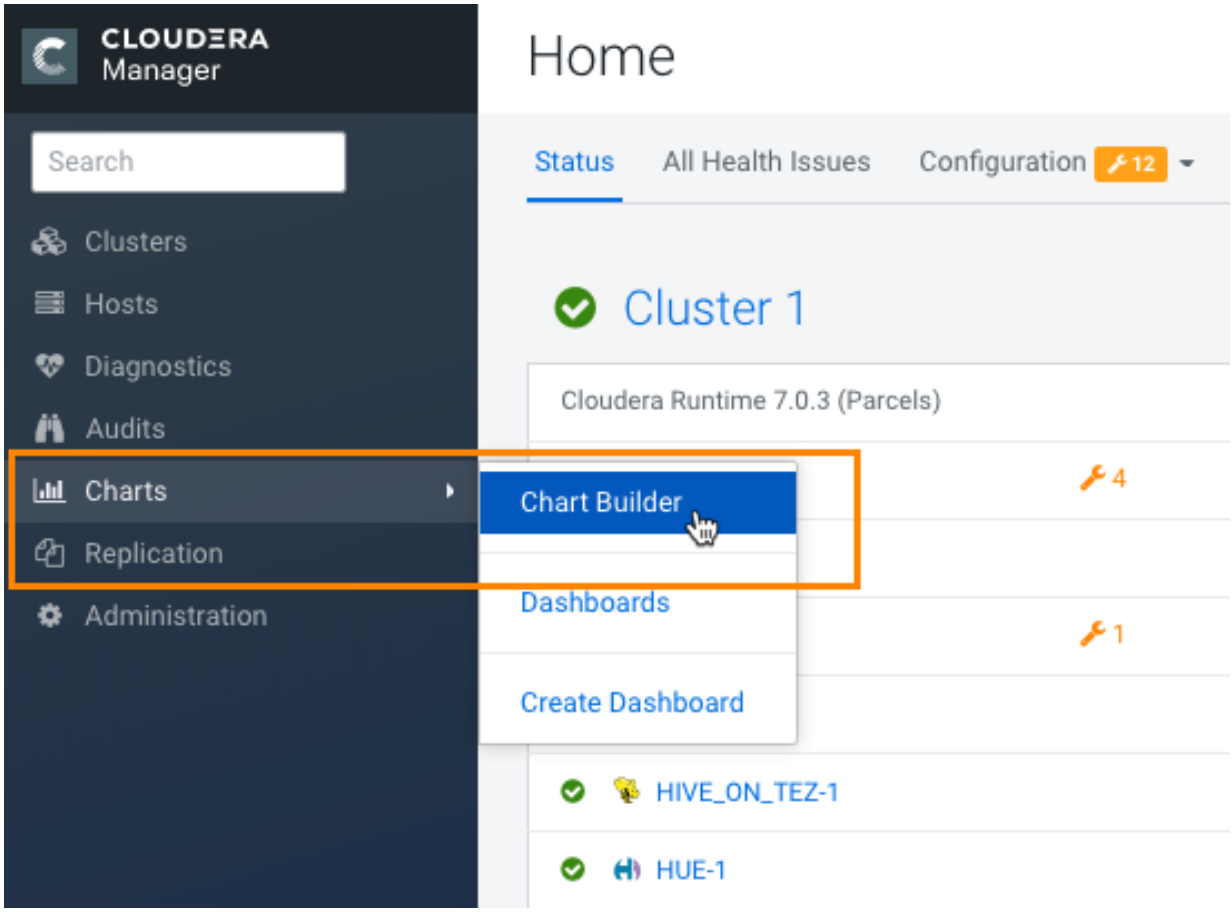
Estimating memory limits

Use Cloudera Manager Chart Builder to determine memory limits for resource pools.

To identify what is the appropriate minimum, maximum, and default memory limit for each of the pools, Cloudera recommends that you run real queries on your datasets, and then review the memory usage during peak periods of this test.

After you have run a set of your real queries on your datasets, use Cloudera Manager Chart Builder to view charts that reflect memory usage:

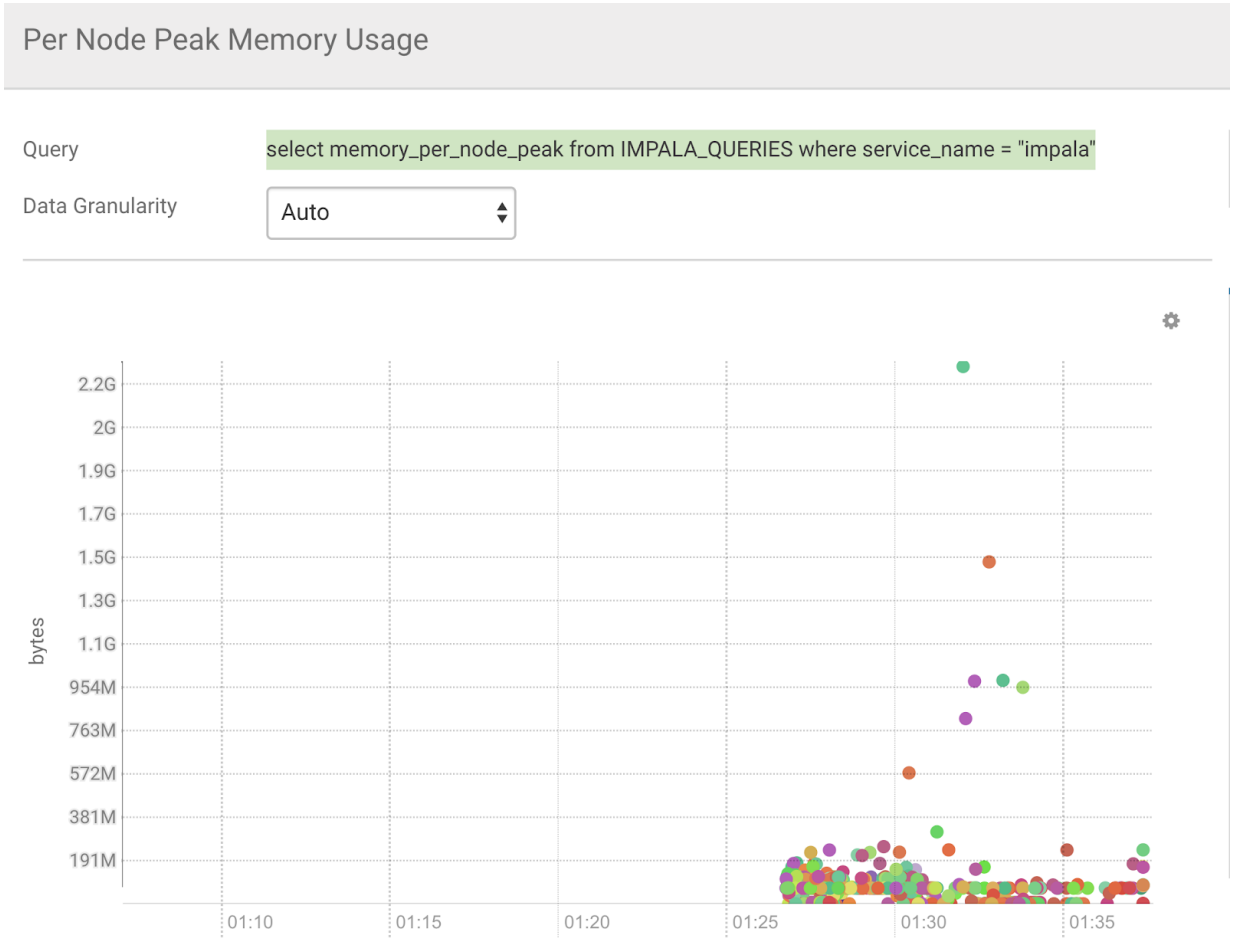
1. On the Cloudera Manager Admin Console home page, from the left navigation tree, select Charts Chart Builder :



2. On the Chart Builder page, enter the following query, and then click Build Chart:

```
select memory_per_node_peak from IMPALA_QUERIES where
service_name="impala"
```

Chart Builder then returns a chart that might look similar to the following example:



This type of chart can be used to determine the memory limits for a particular pool. Note that most queries use less than 1.2 GB per node. See Admission Control and Query Queuing for more information about Impala admission control.

Admission Control and Query Queuing

Admission control is an Impala feature that imposes limits on concurrent SQL queries to avoid resource usage spikes and out-of-memory conditions on busy Cloudera clusters.

The admission control feature lets you set an upper limit on the number of concurrent Impala queries and on the memory used by those queries. Any additional queries are queued until the earlier ones finish, rather than being cancelled or running slowly and causing contention. As other queries finish, the queued queries are allowed to proceed.

You can specify these limits and thresholds for each pool or globally so that you can balance the resource usage and throughput among steady well-defined workloads, rare resource-intensive queries, and ad-hoc exploratory queries.

In addition to the threshold values for currently executing queries, you can place limits on the maximum number of queries that are queued (waiting) and a limit on the amount of time they might wait before returning with an error. These queue settings let you ensure that queries do not wait indefinitely so that you can detect and correct “starvation” scenarios.

Queries, DML statements, and some DDL statements, including CREATE TABLE AS SELECT and COMPUTE STATS are affected by admission control.

On a busy Cloudera cluster, you might find there is an optimal number of Impala queries that run concurrently. For example, when the I/O capacity is fully utilized by I/O-intensive queries, you might not find any throughput benefit in running more concurrent queries. By allowing some queries to run at full speed while others wait, rather than having all queries contend for resources and run slowly, admission control can result in higher overall throughput.

For another example, consider a memory-bound workload such as many large joins or aggregation queries. Each such query could briefly use many gigabytes of memory to process intermediate results. Because Impala by default cancels queries that exceed the specified memory limit, running multiple large-scale queries at once might require re-running some queries that are cancelled. In this case, admission control improves the reliability and stability of the overall workload by only allowing as many concurrent queries as the overall memory of the cluster can accommodate.

Concurrent Queries and Admission Control

One way to limit resource usage through admission control is to set an upper limit on the number of concurrent queries. This is the initial technique you might use when you do not have extensive information about memory usage for your workload. The settings can be specified separately for each dynamic resource pool.

Max Running Queries

Maximum number of concurrently running queries in this pool. The default value is unlimited. (optional)

The maximum number of queries that can run concurrently in this pool. The default value is unlimited. Any queries for this pool that exceed Max Running Queries are added to the admission control queue until other queries finish. You can use Max Running Queries in the early stages of resource management, when you do not have extensive data about query memory usage, to determine if the cluster performs better overall if throttling is applied to Impala queries.

For a workload with many small queries, you typically specify a high value for this setting, or leave the default setting of “unlimited”. For a workload with expensive queries, where some number of concurrent queries saturate the memory, I/O, CPU, or network capacity of the cluster, set the value low enough that the cluster resources are not overcommitted for Impala.

Once you have enabled memory-based admission control using other pool settings, you can still use Max Running Queries as a safeguard. If queries exceed either the total estimated memory or the maximum number of concurrent queries, they are added to the queue.

If Max Running Queries Multiple is set, the Max Running Queries setting is ignored.

Max Running Queries Multiple

This floating point number is multiplied by the current total number of executors at runtime to give the maximum number of concurrently running queries allowed in the pool. The effect of this setting scales with the number of executors in the resource pool.

This calculation is rounded up to the nearest integer, so the result will always be at least one.

If set to zero or a negative number, the setting is ignored.

Max Queued Queries

Maximum number of queries that can be queued in this pool. The default value is 200. (optional).

If Max Queued Queries Multiple is set, the Max Queued Queries setting is ignored.

Max Queued Queries Multiple

This floating point number is multiplied by the current total number of executors at runtime to give the maximum number of queries that can be queued in the pool. The effect of this setting scales with the number of executors in the resource pool.

This calculation is rounded up to the nearest integer, so the result will always be at least one.

If set to zero or a negative number, the setting is ignored.

Queue Timeout

The amount of time, in milliseconds, that a query waits in the admission control queue for this pool before being canceled. The default value is 60,000 milliseconds.

In the following cases, Queue Timeout is not significant, and you can specify a high value to avoid canceling queries unexpectedly:

- In a low-concurrency workload where few or no queries are queued
- In an environment without a strict SLA, where it does not matter if queries occasionally take longer than usual because they are held in admission control

You might also need to increase the value to use Impala with some business intelligence tools that have their own timeout intervals for queries.

In a high-concurrency workload, especially for queries with a tight SLA, long wait times in admission control can cause a serious problem. For example, if a query needs to run in 10 seconds, and you have tuned it so that it runs in 8 seconds, it violates its SLA if it waits in the admission control queue longer than 2 seconds. In a case like this, set a low timeout value and monitor how many queries are cancelled because of timeouts. This technique helps you to discover capacity, tuning, and scaling problems early, and helps avoid wasting resources by running expensive queries that have already missed their SLA.

If you identify some queries that can have a high timeout value, and others that benefit from a low timeout value, you can create separate pools with different values for this setting.

You can combine these settings with the memory-based approach described below. If either the maximum number of or the expected memory usage of the concurrent queries is exceeded, subsequent queries are queued until the concurrent workload falls below the threshold again.

Memory Limits and Admission Control

Each dynamic resource pool can have an upper limit on the cluster-wide memory used by queries executing in that pool.

Use the following settings to manage memory-based admission control.

Max Memory

The maximum amount of aggregate memory available across the cluster to all queries executing in this pool. This should be a portion of the aggregate configured memory for Impala daemons, which will be shown in the settings dialog next to this option for convenience. Setting this to a non-zero value enables memory based admission control.

Impala determines the expected maximum memory used by all queries in the pool and holds back any further queries that would result in Max Memory being exceeded.

If you specify Max Memory, you should specify the amount of memory to allocate to each query in this pool. You can do this in two ways:

- By setting Maximum Query Memory Limit and Minimum Query Memory Limit. This is preferred as it gives Impala flexibility to set aside more memory to queries that are expected to be memory-hungry.
- By setting Default Query Memory Limit to the exact amount of memory that Impala should set aside for queries in that pool.

Note that in the following cases, Impala will rely entirely on memory estimates to determine how much memory to set aside for each query. This is not recommended because it can result in queries not running or being starved for memory if the estimates are inaccurate. And it can affect other queries running on the same node.

- Max Memory, Maximum Query Memory Limit, and Minimum Query Memory Limit are not set, and the MEM_LIMIT query option is not set for the query.
- Default Query Memory Limit is set to 0, and the MEM_LIMIT query option is not set for the query.

If Max Memory Multiple is set, the Max Memory setting is ignored.

Max Memory Multiple

This number of bytes is multiplied by the current total number of executors at runtime to give the maximum memory available across the cluster for the pool. The effect of this setting scales with the number of executors in the resource pool.

If set to zero or a negative number, the setting is ignored.

Minimum Query Memory Limit and Maximum Query Memory Limit

These two options determine the minimum and maximum per-host memory limit that will be chosen by Impala Admission control for queries in this resource pool. If set, Impala admission control will choose a memory limit between the minimum and maximum values based on the per-host memory estimate for the query. The memory limit chosen determines the amount of memory that Impala admission control will set aside for this query on each host that the query is running on. The aggregate memory across all of the hosts that the query is running on is counted against the pool's Max Memory.

Minimum Query Memory Limit must be less than or equal to Maximum Query Memory Limit and Max Memory.

A user can override Impala's choice of memory limit by setting the MEM_LIMIT query option. If the Clamp MEM_LIMIT Query Option setting is set to TRUE and the user sets MEM_LIMIT to a value that is outside of the range specified by these two options, then the effective memory limit will be either the minimum or maximum, depending on whether MEM_LIMIT is lower than or higher the range.

For example, assume a resource pool with the following parameters set:

- Minimum Query Memory Limit = 2GB
- Maximum Query Memory Limit = 10GB

If a user tries to submit a query with the MEM_LIMIT query option set to 14 GB, the following would happen:

- If Clamp MEM_LIMIT Query Option = true, admission controller would override MEM_LIMIT with 10 GB and attempt admission using that value.
- If Clamp MEM_LIMIT Query Option = false, the admission controller will retain the MEM_LIMIT of 14 GB set by the user and will attempt admission using the value.

Clamp MEM_LIMIT Query Option

If this field is not selected, the MEM_LIMIT query option will not be bounded by the Maximum Query Memory Limit and the Minimum Query Memory Limit values specified for this resource pool. By default, this field is selected. The field is disabled if both Minimum Query Memory Limit and Maximum Query Memory Limit are not set.

You can combine the memory-based settings with the upper limit on concurrent queries. If either the maximum number of or the expected memory usage of the concurrent queries is exceeded, subsequent queries are queued until the concurrent workload falls below the threshold again.

Monitoring Admission Control

To see how admission control works for particular queries, examine the profile output or the summary output for the query.

Profile

The information is available through the PROFILE statement in `impala-shell` immediately after running a query in the shell, on the queries page of the Impala debug web UI, or in the Impala log file (basic information at log level 1, more detailed information at log level 2).

The profile output contains details about the admission decision, such as whether the query was queued or not and which resource pool it was assigned to. It also includes the estimated and actual memory usage for the query, so you can fine-tune the configuration for the memory limits of the resource pools.

Summary

The summary output includes the queuing status consisting of whether the query was queued and what was the latest queuing reason.

The information is available in `impala-shell` when the LIVE_PROGRESS or LIVE_SUMMARY query option is set to TRUE.

You can also start an impala-shell session with the --live_progress or --live_summary flags to monitor all queries in that impala-shell session.

Enabling Admission Control

Enable admission control on all production clusters to alleviate possible capacity issues. The capacity issues could be because of a high volume of concurrent queries, because of heavy-duty join and aggregation queries that require large amounts of memory, or because Impala is being used alongside other Hadoop data management components and the resource usage of Impala must be constrained to work well.

About this task

Procedure

1. Navigate to ClustersImpala.
2. In the Configuration tab, navigate to CategoryAdmission Control.
3. Select or clear both the Enable Impala Admission Control and the Enable Dynamic Resource Pools.
4. Enter a Reason for change, and then click Save Changes to commit the changes.
5. Restart the Impala service.
6. After completing this task, for further configuration settings, customize the configuration settings for the dynamic resource pools.

Creating Static Pools

To manage and prioritize workloads on clusters, use the static service pools to allocate dedicated resources for Impala for predictable resource availability. When static service pools are used, Cloudera Manager creates a cgroup in which Impala runs. This cgroup limits memory, CPU and Disk I/O according to the static partitioning policy.

About this task

Create static resource pools for the services running in your Cloudera cluster.

Procedure

1. In Cloudera Manager, navigate to Clusters Static service pools .
2. In the Configuration tab, allocate a portion of resource to each component.
   - HDFS always needs to have a minimum of 5-10% of the resources.
   - Generally, YARN and Impala split the rest of the resources.
     - For mostly batch workloads, you might allocate YARN 60%, Impala 30%, and HDFS 10%.
     - For mostly ad-hoc query workloads, you might allocate Impala 60%, YARN 30%, and HDFS 10%.
3. Click Continue.

4. Review the changes and click Continue.
5. Click Restart Now.

Configuring Dynamic Resource Pool

Admission control and dynamic resource pools are enabled by default. However, until you configure the settings for the dynamic resource pools, the admission control feature is effectively not enabled.

About this task

There is always a resource pool designated as root.default. By default, all Impala queries run in this pool when the dynamic resource pool feature is enabled for Impala. You create additional pools when your workload includes identifiable groups of queries (such as from a particular application, or a particular group within your organization) that have their own requirements for concurrency, memory use, or service level agreement (SLA). Each pool has its own settings related to memory, number of queries, and timeout interval.

Procedure

1. In Cloudera Manager, navigate to Clusters Impala Admission Control Configuration . If the cluster has an Impala service, the Resource Pools tab displays under the Impala Admission Control tab.
2. In the Impala Admission Control tab, click Create Resource Pool.
3. Specify a name and resource limits for the pool:
   - In the Resource Pool Name field, type a unique name containing only alphanumeric characters.
   - Optionally, in the Submission Access Control tab, specify which users and groups can submit queries. By default, anyone can submit queries. To restrict this permission, select the Allow these users and groups option and provide a comma-delimited list of users and groups in the Users and Groups fields respectively.
4. Click Create.
5. Click Refresh Dynamic Resource Pools.

Dynamic Resource Pool Settings

Use the following settings to configure your dynamic resource pools for Impala.

Max Memory

Maximum amount of aggregate memory available across the cluster to all queries executing in this pool. This should be a portion of the aggregate configured memory for Impala daemons, which will be shown in the settings dialog next to this option for convenience. Setting this to a non-zero value enables memory based admission control.

Impala determines the expected maximum memory used by all queries in the pool and holds back any further queries that would result in Max Memory being exceeded.

If you specify Max Memory, you should specify the amount of memory to allocate to each query in this pool. You can do this in two ways:

- By setting Maximum Query Memory Limit and Minimum Query Memory Limit. Setting them gives Impala flexibility to set aside more memory to queries that are expected to be memory-hungry.
- By setting Default Query Memory Limit to the exact amount of memory that Impala should set aside for queries in that pool.

Note that if you do not set any of the above options, or set Default Query Memory Limit to 0, Impala will rely entirely on memory estimates to determine how much memory to set aside for each query. This is not recommended because it can result in queries not running or being starved for memory if the estimates are inaccurate.

For example, consider the following scenario:

- The cluster is running impalad daemons on five hosts.
- A dynamic resource pool has Max Memory set to 100 GB.
- The Maximum Query Memory Limit for the pool is 10 GB and Minimum Query Memory Limit is 2 GB. Therefore, any query running in this pool could use up to 50 GB of memory (Maximum Query Memory Limit * number of Impala nodes).
- Impala will run varying numbers of queries concurrently because queries may be given memory limits anywhere between 2 GB and 10 GB, depending on the estimated memory requirements. For example, Impala may run up to 10 small queries with 2 GB memory limits or two large queries with 10 GB memory limits because that is what will fit in the 100 GB cluster-wide limit when executing on five hosts.
- The executing queries may use less memory than the per-host memory limit or the Max Memory cluster-wide limit if they do not need that much memory. In general this is not a problem so long as you are able to run enough queries concurrently to meet your needs.

Minimum Query Memory Limit and Maximum Query Memory Limit

These two options determine the minimum and maximum per-host memory limit that will be chosen by Impala Admission control for queries in this resource pool. If set, Impala admission control will choose a memory limit between the minimum and maximum value based on the per-host memory estimate for the query. The memory limit chosen determines the amount of memory that Impala admission control will set aside for this query on each host that the query is running on. The aggregate memory across all of the hosts that the query is running on is counted against the pool's Max Memory.

Minimum Query Memory Limit must be less than or equal to Maximum Query Memory Limit and Max Memory.

You can override Impala's choice of memory limit by setting the MEM_LIMIT query option. If the Clamp MEM_LIMIT Query Option is selected and the user sets MEM_LIMIT to a value that is outside of the range specified by these two options, then the effective memory limit will be either the minimum or maximum, depending on whether MEM_LIMIT is lower than or higher than the range.

Max Running Queries

Maximum number of concurrently running queries in this pool. The default value is unlimited.

The maximum number of queries that can run concurrently in this pool. The default value is unlimited. Any queries for this pool that exceed Max Running Queries are added to the admission control queue until other queries finish. You can use Max Running Queries in the early stages of resource management, when you do not have extensive data about query memory usage, to determine if the cluster performs better overall if throttling is applied to Impala queries.

For a workload with many small queries, you typically specify a high value for this setting, or leave the default setting of "unlimited". For a workload with expensive queries, where some number of concurrent queries saturate the memory, I/O, CPU, or network capacity of the cluster, set the value low enough that the cluster resources are not overcommitted for Impala.

Once you have enabled memory-based admission control using other pool settings, you can still use Max Running Queries as a safeguard. If queries exceed either the total estimated memory or the maximum number of concurrent queries, they are added to the queue.

Max Queued Queries

Maximum number of queries that can be queued in this pool. The default value is 200. (optional)

Queue Timeout

The amount of time, in milliseconds, that a query waits in the admission control queue for this pool before being canceled. The default value is 60,000 milliseconds.

It the following cases, Queue Timeout is not significant, and you can specify a high value to avoid canceling queries unexpectedly:

- In a low-concurrency workload where few or no queries are queued
- In an environment without a strict SLA, where it does not matter if queries occasionally take longer than usual because they are held in admission control

You might also need to increase the value to use Impala with some business intelligence tools that have their own timeout intervals for queries.

In a high-concurrency workload, especially for queries with a tight SLA, long wait times in admission control can cause a serious problem. For example, if a query needs to run in 10 seconds, and you have tuned it so that it runs in 8 seconds, it violates its SLA if it waits in the admission control queue longer than 2 seconds. In a case like this, set a low timeout value and monitor how many queries are cancelled because of timeouts. This technique helps you to discover capacity, tuning, and scaling problems early, and helps avoid wasting resources by running expensive queries that have already missed their SLA.

If you identify some queries that can have a high timeout value, and others that benefit from a low timeout value, you can create separate pools with different values for this setting.

Clamp MEM_LIMIT Query Option

If this field is not selected, the MEM_LIMIT query option will not be bounded by the Maximum Query Memory Limit and the Minimum Query Memory Limit values specified for this resource pool. By default, this field is selected. The field is disabled if both Minimum Query Memory Limit and Maximum Query Memory Limit are not set.

Admission Control Sample Scenario

You can learn about the factors you must consider when allocating Impala's resources and the process you need to follow to set up admission control for the selected workload.

Anne Chang is administrator for an enterprise data hub that runs a number of workloads, including Impala.

Anne has a 20-node cluster that uses Cloudera Manager static partitioning. Because of the heavy Impala workload, Anne needs to make sure Impala gets enough resources. While the best configuration values might not be known in advance, she decides to start by allocating 50% of resources to Impala. Each node has 128 GiB dedicated to each impalad. Impala has 2560 GiB in aggregate that can be shared across the resource pools she creates.

Next, Anne studies the workload in more detail. After some research, she might choose to revisit these initial values for static partitioning.

To figure out how to further allocate Impala's resources, Anne needs to consider the workloads and users, and determine their requirements. There are a few main sources of Impala queries:

- Large reporting queries executed by an external process/tool. These are critical business intelligence queries that are important for business decisions. It is important that they get the resources they need to run. There typically are not many of these queries at a given time.
- Frequent, small queries generated by a web UI. These queries scan a limited amount of data and do not require expensive joins or aggregations. These queries are important, but not as critical, perhaps the client tries resending the query or the end user refreshes the page.
- Occasionally, expert users might run ad-hoc queries. The queries can vary significantly in their resource requirements. While Anne wants a good experience for these users, it is hard to control what they do (for example, submitting inefficient or incorrect queries by mistake). Anne restricts these queries by default and tells users to reach out to her if they need more resources.

To set up admission control for this workload, Anne first runs the workloads independently, so that she can observe the workload's resource usage in Cloudera Manager. If they could not easily be run manually, but had been run in the past, Anne uses the history information from Cloudera Manager. It can be helpful to use other search criteria (for example, user) to isolate queries by workload. Anne uses the Cloudera Manager chart for Per-Node Peak Memory usage to identify the maximum memory requirements for the queries.

From this data, Anne observes the following about the queries in the groups above:

- Large reporting queries use up to 32 GiB per node. There are typically 1 or 2 queries running at a time. On one occasion, she observed that 3 of these queries were running concurrently. Queries can take 3 minutes to complete.
- Web UI-generated queries use between 100 MiB per node to usually less than 4 GiB per node of memory, but occasionally as much as 10 GiB per node. Queries take, on average, 5 seconds, and there can be as many as 140 incoming queries per minute.
- Anne has little data on ad hoc queries, but some are trivial (approximately 100 MiB per node), others join several tables (requiring a few GiB per node), and one user submitted a huge cross join of all tables that used all system resources (that was likely a mistake).

Based on these observations, Anne creates the admission control configuration with the following pools:

XL_Reporting

Property	Value
Max Memory	1280 GiB
Maximum Query Memory Limit	32 GiB
Minimum Query Memory Limit	32 GiB
Max Running Queries	2
Queue Timeout	5 minutes

This pool is for large reporting queries. To support running 2 queries at a time, the pool memory resources are set to 1280 GiB (aggregate cluster memory). This is for 2 queries, each with 32 GiB per node, across 20 nodes. Anne sets the pool's Maximum Query Memory Limit to 32 GiB so that no query uses more than 32 GiB on any given node. She sets Max Running Queries to 2 (though it is not necessary she do so). She increases the pool's queue timeout to 5 minutes in case a third query comes in and has to wait. She does not expect more than 3 concurrent queries, and she does not want them to wait that long anyway, so she does not increase the queue timeout. If the workload increases in the future, she might choose to adjust the configuration or buy more hardware.

HighThroughput_UI

Property	Value
Max Memory	960 GiB (inferred)
Maximum Query Memory Limit	4 GiB
Minimum Query Memory Limit	2 GiB
Max Running Queries	12
Queue Timeout	5 minutes

This pool is used for the small, high throughput queries generated by the web tool. Anne sets the Maximum Query Memory Limit to 4 GiB per node, and sets Max Running Queries to 12. This implies a maximum amount of memory per node used by the queries in this pool: 48 GiB per node (12 queries * 4 GiB per node memory limit).

Notice that Anne does not set the pool memory resources, but does set the pool's Maximum Query Memory Limit. This is intentional: admission control processes queries faster when a pool uses the Max Running Queries limit instead of the peak memory resources.

This should be enough memory for most queries, since only a few go over 4 GiB per node. For those that do require more memory, they can probably still complete with less memory (spilling if necessary). If, on occasion, a query cannot run with this much memory and it fails, Anne might reconsider this configuration later, or perhaps she does not need to worry about a few rare failures from this web UI.

With regard to throughput, since these queries take around 5 seconds and she is allowing 12 concurrent queries, the pool should be able to handle approximately 144 queries per minute, which is enough for the peak maximum expected of 140 queries per minute. In case there is a large burst of queries, Anne wants them to queue. The default maximum size of the queue is already 200, which should be more than large enough. Anne does not need to change it.

Default

Property	Value
Max Memory	320 GiB
Maximum Query Memory Limit	4 GiB
Minimum Query Memory Limit	2 GiB
Max Running Queries	Unlimited
Queue Timeout	60 Seconds

The default pool (which already exists) is a catch all for ad-hoc queries. Anne wants to use the remaining memory not used by the first two pools, 16 GiB per node (XL_Reporting uses 64 GiB per node, High_Throughput_UI uses 48 GiB per node). For the other pools to get the resources they expect, she must still set the Max Memory resources and the Maximum Query Memory Limit. She sets the Max Memory resources to 320 GiB (16 * 20). She sets the Maximum Query Memory Limit to 4 GiB per node for now. That is somewhat arbitrary, but satisfies some of the ad hoc queries she observed. If someone writes a bad query by mistake, she does not actually want it using all the system resources. If a user has a large query to submit, an expert user can override the Maximum Query Memory Limit (up to 16 GiB per node, since that is bound by the pool Max Memory resources). If that is still insufficient for this user's workload, the user should work with Anne to adjust the settings and perhaps create a dedicated pool for the workload.

Cancelling a Query

Various client applications let you interactively cancel queries submitted or monitored through those applications.

Setting a Time Limit on Query Execution

An Impala administrator can set a default value of the EXEC_TIME_LIMIT_S query option for a resource pool. If a user accidentally runs a large query that executes for longer than the limit, it will be automatically terminated after the time limit expires to free up resources.

You can override the default value per query or per session if you do not want to apply the default EXEC_TIME_LIMIT_S value to a specific query or a session.

Interactively Cancelling a Query

- In the Impala Web UI for the `impalad` host (on port 25000 by default), cancel a query: In the /queriestab, click Cancel for a query in the queries in flight list.
- In `impala-shell`, press ^C
- In Data Explorer (Hue), click Cancel from the Watch page.

You can manually cancel a query in the Impala Web UI for the `impalad` host (on port 25000 by default):

User quotas in Admission Control (Preview)

User quotas in Impala's Admission Control set per-user query limits to ensure fair resource distribution and prevent system overload.

Note: This feature is in technical preview and not recommended for use in production deployments. Cloudera recommends that you try this feature in test and development environments.

Starting with Cloudera Runtime 7.3.1.500 SP3, user quotas introduce rules to restrict the number of queries a users/groups can run concurrently. These rules apply at both the pool and root levels and can be based on individual usernames, wildcard users, or user groups. Queries are counted against limits starting with admission control acceptance and continuing until they are released.

When a query exceeds the defined quota, it is rejected at submission time.

Note: Note: The query counts are synchronized across coordinators through the Statestore, which may lead to over-admission.

User quota configuration elements

List of XML elements used to configure user quotas in Impala's Admission Control.

Element	Element
userQueryLimit	Used to define a User or Wildcard Rule
totalCount	Used to define a Group Rule
totalCount	Used to define the number of queries that can run concurrently.
user	Used to specify a username to define a User Rule, or, by using the wildcard '*', to define a Wildcard Rule.
group	In a Group rule, used to specify a group name that the rule applies to.

Rule precedence:

- User rules override group and wildcard rules
- Group rules override wildcard rules.
- Pool-level rules are evaluated first; if passed, root-level rules are checked.
- If a user belongs to multiple groups, the least restrictive rule applies.

Configuring user quotas in Admission Control

Configure user quotas in Impala's Admission Control to limit concurrent queries for individual users or groups.

Before you begin

Ensure you have access to the fair-scheduler.xml file and necessary administrative privileges in Impala.

Procedure

1. Log in to Cloudera Manageras an Administrator
2. In Cloudera Manager, go to Impala Configurations Impala Daemon Fair Scheduler Advanced Configuration Snippet (Safety Valve) .
3. Add or Update <userQueryLimit> and <groupQueryLimit> elements to define the quota rules.

```
<queue name="group-set-small">
 <userQueryLimit>
 <user>*</user>
 <totalCount>1</totalCount>
 </userQueryLimit>
 <groupQueryLimit>
 <group>it</group>
 <totalCount>2</totalCount>
 </groupQueryLimit>
 <userQueryLimit>
 <user>fiona</user>
 <totalCount>3</totalCount>
 </userQueryLimit>
</queue>
```

4. Click **Apply Changes** button and restart Impala.

Example configuration rules

Example rule definitions for user, group, and wildcard quotas.

The examples below are incomplete XML snippets used within Admission Control configuration files. They omit required elements such as <aclSubmitApps>.

Example:

```
<queue name="group-set-small">
<!-- Note: for brevity's sake, this example intentionality excludes other
 elements such
as weight, schedulingPolicy, and aclSubmitApps -->
<!-- Any user can run 1 query in the small pool -->
<userQueryLimit>
<user>*</user>
<totalCount>1</totalCount>
</userQueryLimit>
<!-- Members of the group 'it' can run 2 queries in the small pool -->
<groupQueryLimit>
<group>it</group>
<totalCount>2</totalCount>
</groupQueryLimit>
<!-- The user 'fiona' can run 3 queries in the small pool -->
<userQueryLimit>
<user>fiona</user>
<totalCount>3</totalCount>
</userQuery
```

In this example:

- Any user can run 1 query (* wildcard rule).
- Users in the it group can run 2 queries
- The user fiona can run 3 queries, overriding the it group rule.

User quotas help administrators ensure resource fairness and prevent system overloads by restricting query concurrency at both user and group levels. For more configuration examples, see *Impala documentation on user quotas*.

Related Information

Impala documentation on user quotas

Limitation

This feature is not supported for virtual warehouses that use workload-aware autoscaling

When an Impala virtual warehouse has been auto-suspended, you can submit as many queries as desired. All these queries will be queued with the message:

```
Query queued. Latest queuing reason: Waiting for executors to start.
Only DDL queries and queries scheduled only on the coordinator
(either NUM_NODES set to 1 or when small query optimization is
triggered) can currently run.
```

Once the virtual warehouse resumes, all queued queries will be admitted and executed simultaneously. The limit defined in the llama.am.throttling.maximum.queued.reservations setting in llama-site.xml still applies. These limitations are due to a known issue tracked at IMPALA-13965. There is currently no workaround for these issues.

Query counts are synchronized across the cluster by using the statestore. Because the statestore uses an eventually consistent model, the system may admit more queries than the configured limit when multiple queries start at the same time.

Using HLL Datasketch Algorithms in Impala

You can use Datasketch algorithms (HLL) for queries that take too long to calculate exact results due to very large data sets (e.g. number of distinct values).

You may use data sketches (HLL algorithms) to generate approximate results that are much faster to retrieve. HLL is an algorithm that gives approximate answers for computing the number of distinct values in a column. The value returned by this algorithm is similar to the result of COUNT(DISTINCT col) and the NDV function integrated with Impala. However, HLL algorithm is much faster than COUNT(DISTINCT col) and the NDV function and is less memory-intensive for columns with high cardinality.

These HLL algorithms create “sketches”, which are data structures containing an approximate calculation of some facet of the data and which are compatible regardless of which system reads or writes them. For e.g., This particular algorithm works in two phases. First it creates a sketch from the dataset provided for the algorithm and then as the second step in the process you can read the estimate from the sketch. Using this approach it’s also possible to create granular sketches for a specific dataset (e.g. one sketch per partition) and later on use them to provide answers for different count(distinct) queries without reading the actual data as they can also be merged together. Using HLL you can create sketches by Hive and read it with Impala for getting the estimate and vice versa.

Usage Notes

Query results produced by the HLL algorithm are approximate but within well defined error bounds. Because the number of distinct values in a column returned by this HLL algorithm is an estimate, it might not reflect the precise number of different values in the column, especially if the cardinality is very high.

Data Types

The following is a list of currently supported data types of the datasets that these HLL functions are compatible with.

Supported:

1. Numeric Types:

   TINYINT, INT, BIGINT, FLOAT, DOUBLE
2. String types:

   STRING, CHAR, VARCHAR
3. Other types:

   BINARY

Unsupported:

1. Complex types
2. Some of the scalar types: DECIMAL, TIMESTAMP, BOOLEAN, SMALLINT, DATE

File Formats

The currently supported file formats are ORC and Parquet. Since Impala does not yet have read support for materialized views, it is required for Hive to write the sketches into regular tables (i.e. not into a materialized view), using the BINARY type. Then Impala can be used to read these sketches and calculate the estimate as Hive does.

Limitations:

You can write HLL sketches into a text table, however it may not work as the serialized sketches can contain characters that are special for text format.

Internal Configuration

The configuration is hard coded and the level of compression is HLL_4 and the number of buckets in the HLL array is k=12 (2 power of 12).

Using HLL Sketches

This datasketch algorithm works in two phases.

- First it creates a sketch from the dataset provided for the algorithm.
- As the second step in the process the function provides an estimate from the sketch.

The following example shows how to create a sketch and to create an estimate from the sketch.

Creating a sketch

This query creates a sketch using the ds_hll_sketch function from a column date_string_col containing the supported data type.

```
Select ds_hll_sketch(date_string_col) from tableName;
```

This query receives a primitive expression and returns a serialized HLL sketch that is not in a readble format.

Estimating the number of distinct values in a column

The following query returns a cardinality estimate (similarly to ndv()) for that particular column (date_string_col) by using the HLL function ds_hll_estimate.

```
Select ds_hll_estimate(ds_hll_sketch(date_string_col)) from tableName;
```

This query receives a primitive expression (column name) and then creates a sketch from the column, and then passes the sketch to the outer function that gives a cardinality estimate.

```
+-------------------------------------------------+
| ds_hll_estimate(ds_hll_sketch(date_string_col))|
+-------------------------------------------------+
| 729                                             |
+-------------------------------------------------+
Fetched 1 row(s) in 0.25s
```

The above result is similar to the results received using the count(distinct col) or the ndv() approach.

```
Select count(distinct date_string_col) from tableName;
+-------------------------------------------------+
| count (distinct date_string_col)                |
+-------------------------------------------------+
| 730                                             |
+-------------------------------------------------+
```

```
Select ndv(distinct date_string_col) from tableName;
+-------------------------------------------------+
| ndv (date_string_col)                           |
+-------------------------------------------------+
| 736                                             |
+-------------------------------------------------+
```

Inserting the derived sketch into a table

You can create a sketch and save it for later use. If you save the sketches to a table or view with the same granularity (e.g. one sketch for each partition in a table) then later you can simply read the sketch and get the estimate almost for free as the real cost here is just the sketch creation.

The following example shows the creation of a sketch for the column (date_string_col) from the table “tableName” and saving the created sketch into another table called sketch_table and later reading the sketch and its estimate from the saved table called sketch_table.

```
    select year, month, ds_hll_estimate(ds_hll_sketch(date_string_col)) from
 tableName group by year, month;
    +-------------------------------------------------------------------+
    | year | month | ds_hll_estimate(ds_hll_sketch(date_string_col))|
    +------+-------+-------------------------------------------------+
    | 2010 | 5     | 31                                              |
    | 2010 | 11    | 30                                              |
    | 2010 | 6     | 30                                              |
    | 2010 | 2     | 28                                              |
    | 2010 | 10    | 31                                              |
    | 2009 | 11    | 30                                              |
    | 2009 | 7     | 31                                              |
    | 2009 | 10    | 31                                              |
    | 2010 | 11    | 31                                              |
    | 2009 | 7     | 31                                              |
    | 2010 | 10    | 31                                              |
    | 2010 | 8     | 30                                              |
    | 2010 | 5     | 31                                              |
    | 2009 | 7     | 30                                              |
    | 2010 | 4     | 31                                              |
    | 2010 | 12    | 30                                              |
    | 2009 | 9     | 31                                              |
    | 2009 | 8     | 30                                              |
    | 2010 | 6     | 28                                              |
    | 2010 | 3     | 31                                              |
    | 2010 | 4     | 31                                              |
    | 2010 | 2     | 30                                              |
    | 2009 | 1     | 31                                              |
    | 2009 | 12    | 30                                              |
    +-------------------------------------------------------------------+
```

Insert the created sketch into another table:

```
    insert into sketch_table select year, month, ds_hll_sketch(date_string_
 col) from tableName group by year, month;
```

Verify the number of rows modified using count function in the table:

```
    select count(1) from sketch_table;
```

You can also verify it by getting the estimates from the sketches stored in the table using the following query:

```
    select year, month, ds_hll_estimate(sketch_col) from sketch_table;
```

Combining Sketches

You can combine sketches to allow complex queries to be computed from a few number of simple sketches.

```
select ds_hll_estimate(ds_hll_union(sketch_col)) from sketch_table;
```

Non-deterministic vs Deterministic Results

Based on the characteristics of a query, DataSketches HLL might return results in a non-deterministic manner. During the cache populating phase it returns exact results but in the approximating phase the algorithm can be sensitive to the order of data it receives. As a result you might see the results differ in consecutive runs but all of them will be within the error range of the algorithm.

Due to the characteristics of the algorithm, and to how it was implemented in Impala, the results might be non-deterministic when the query is executed on one node and the input data set is big enough that the algorithm starts approximating instead of giving exact results.

Inter-opreability between Hive and Impala

From the above examples you will notice that the NDV function is much faster than the Datasketches. But NDV is used only by Impala and not by Hive whereas the Datasketches are implemented in Hive as well as in Impala. In order to maintain the inter-operability between Hive and Impala we recommend to use Datasketches to calculate the estimate. A sketch created by Hive using Datasketches can be fed to Impala to produce an estimate.

Using KLL Datasketch Algorithms in Impala

You can use the stochastic streaming algorithm (KLL) that uses the percentile/quantile functions to statistically analyze the approximate distribution of comparable data from a very large stream.

Use the ds_kll_quantiles() or its inverse functions the Probability Mass Function ds_kll_PMF() and the Cumulative Distribution Function ds_kll_CDF() to obtain the analysis. Query results produced by this KLL algorithm are approximate but within well defined error bounds.

Data Types and File Formats

KLL function currently supports only floats. If you sketch an int data the KLL function converts it into float.

You can sketch data from any file formats. However it is recommended to save the sketches into parquet tables. It is not recommended to save the sketches to text format.

Using KLL Sketches

This datasketch algorithm works in two phases.

- First it creates a sketch from the dataset provided for the algorithm.
- As the second step in the process you can read the estimate from the sketch.

Using this approach you can create granular sketches for a specific dataset (e.g. one sketch per partition) and later use them to provide answers for different quantile queries without reading the actual data as they can also be merged together. Using KLL you can create sketches by Hive and read it with Impala for getting the estimate and vice versa.

Creating a sketch

The following example shows how to create a sketch using the ds_kll_sketch function from a column (float_col) containing the supported data type. This created sketch is a data structure containing an approximate calculation of some facet of the data and which are compatible regardless of which system reads or writes them.

```
select ds_kll_sketch(float_col) from tableName;
```

where ds_kll_sketch() is an aggregate function that receives a float parameter (e.g. a float column of a table) and returns a serialized Apache DataSketches KLL sketch of the input data set wrapped into STRING type. Since the serialized data might contain unsupported characters this query may error out. So another approach to this method is to insert the derived sketch into a table or a view and later use for quantile approximations.

Inserting the derived sketch into a table

If you save the sketches to a table or view with the same granularity (e.g. one sketch for each partition in a table) then later you can simply read the sketch for quantile approximation.

```
insert into table_name select ds_kll_sketch(float_col) from tableName;
```

Debugging the created sketch

To troubleshoot the sketch created by the ds_kll_sketch function, use ds_kll_stringify on the sketch. ds_kll_stringify () receives a string that is a serialized Apache DataSketches sketch and returns its stringified format by invoking the related function on the sketch's interface.

```
select ds_kll_stringify(ds_kll_sketch(float_col)) from tableName;
```

```
+-------------------------------------------+
| ds_kll_stringify(ds_kll_sketch(float_col))|
+-------------------------------------------+
| ### KLL sketch summary:                   |
|    K                     : 200            |
|    min K                 : 200            |
|    M                     : 8              |
|    N                     : 100            |
|    Epsilon               : 1.33%          |
|    Epsilon PMF           : 1.65%          |
|    Empty                 : false          |
|    Estimation mode       : false          |
|    Levels                : 1              |
|    Sorted                : false          |
|    Capacity items        : 200            |
|    Retained items        : 100            |
|    Storage bytes         : 432            |
|    Min value             : 0              |
|    Max value             : 9              |
| ### End sketch summary                    |
+-------------------------------------------+
```

Determining the number of datasets sketched

To determine the number of records sketched into this sketch, use ds_kll_n function.

```
select ds_kll_n(ds_kll_sketch(float_col)) from tableName;
```

```
+-----------------------------------+
| ds_kll_n(ds_kll_sketch(float_col)) |
+-----------------------------------+
| 100                               |
```

```
    +-----------------------------------+
```

Calculate Quantile

This function ds_kll_quantile() function receives two parameters, a STRING parameter that contains a serialized KLL sketch and a DOUBLE (0.5) that represents the rank of the quantile in the range of [0,1]. E.g. rank=0.5 means the approximate median of all the sketched data.

```
    +-------------------------------------------------+
    | ds_kll_quantile(ds_kll_sketch(float_col), 0.5) |
    +-------------------------------------------------+
    | 4.0                                             |
    +-------------------------------------------------+
```

This query returns a data (4.0) that is bigger than the 50% of the data.

Calculating quantiles

To calculate the quantiles for multiple rank parameters, use the function ds_kll_quantiles_as_string () that is very similar to ds_kll_quantile() but this function receives any number of rank parameters and returns a comma separated string that holds the results for all of the given ranks.

```
   select ds_kll_quantiles_as_string(ds_kll_sketch(float_col), 0.5, 0.6, 1)
from tableName;
```

```
    +-----------------------------------------------------------------------+
    | ds_kll_quantiles_as_string(ds_kll_sketch(float_col), 0.5, 0.6, 1) |
    +-----------------------------------------------------------------------+
    | 4, 5, 9                                                               |
    +-----------------------------------------------------------------------+
```

Calculate Rank

This rank function ds_kll_rank() receives two parameters, a STRING that represents a serialized DataSketches KLL sketch and a float to provide a probing value in the sketch.

```
    select ds_kll_rank(ds_kll_sketch(float_col), 4) from tableName;
```

```
    +-------------------------------------------------+
    | ds_kll_rank(ds_kll_sketch(float_col), 4)       |
    +-------------------------------------------------+
    | 0.48                                            |
    +-------------------------------------------------+
```

This query returns a DOUBLE that is the rank of the given probing value in the range of [0,1]. E.g. a return value of 0.48 means that the probing value given as parameter is greater than 48% of all the values in the sketch.

Note: This is only an approximate calculation.

Calculate Probability Mass Function (PMF)

This Probabilistic Mass Function (PMF) ds_kll_pmf_as_string()receives a serialized KLL sketch and one or more float values to represent ranges in the sketched values. In the following example, the float values [1, 3, 4, 8] represent the following ranges: (-inf, 1), [1, 3), [3, 4), [4, 8) [8, +inf)

Note: The input values for the ranges have to be unique and monotonically increasing.

```
select ds_kll_pmf_as_string(ds_kll_sketch(float_col), 1, 3, 4, 8) from table
Name
```

```
+-----------------------------------------------------------+
| ds_kll_pmf_as_string(ds_kll_sketch(float_col), 1, 3, 4, 8) |
+-----------------------------------------------------------+
| 0.12, 0.24, 0.12, 0.36, 0.16                              |
+-----------------------------------------------------------+
```

This query returns a comma separated string where each value in the string is a number in the range of [0,1] and shows what percentage of the data is in the particular ranges.

Calculate Cumulative Distribution Function (CDF)

This Cumulative Distribution Function (CDF) ds_kll_cmf_as_string()receives a serialized KLL sketch and one or more float values to represent ranges in the sketched values. In the following example, the float values [1, 3, 4, 8] represents the following ranges: (-inf, 1), (-inf, 3), (-inf, 4), (-inf, 8), (-inf, +inf)

Note: The input values for the ranges have to be unique and monotonically increasing.

```
select ds_kll_cdf_as_string(ds_kll_sketch(float_col), 1, 3, 4, 8) from table
Name;
```

```
+-----------------------------------------------------------+
| ds_kll_cdf_as_string(ds_kll_sketch(float_col), 1, 3, 4, 8) |
+-----------------------------------------------------------+
| 0.12, 0.36, 0.48, 0.84, 1                                 |
+-----------------------------------------------------------+
```

This query returns a comma separated string where each value in the string is a number in the range of [0,1] and shows what percentage of the data is in the particular ranges.

Calculate the Union

To take advantage of the UNION function, create granular sketches for a specific dataset (one sketch per partition), write these sketches to a separate table and based on the partition you are interested in, you can UNION the relevant sketches together to get an estimate.

```
insert into sketch_tbl select ds_kll_sketch(float_col) from tableName;
```

```
select ds_kll_quantile(ds_kll_union(sketch_col), 0.5) from sketch_tbl
    where partition_col=1 OR partition_col=5;
```

This function ds_kll_union() receives a set of serialized Apache DataSketches KLL sketches produced by ds_kll_sketch() and merges them into a single sketch.

Managing Metadata in Impala

This section describes various knobs you can use to control how Impala manages its metadata in order to improve performance and scalability.

On-demand Metadata

With the on-demand metadata feature, the Impala coordinators pull metadata as needed from catalogd and cache it locally. The cached metadata gets evicted automatically under memory pressure.

The granularity of on-demand metadata fetches is at the partition level between the coordinator and catalogd. Common use cases like add/drop partitions do not trigger unnecessary serialization/deserialization of large metadata.

The feature can be used in either of the following modes.

Metadata on-demand mode

In this mode, all coordinators use the metadata on-demand.

Set the following on catalogd:

```
--catalog_topic_mode=minimal
```

Set the following on all impalad coordinators:

```
--use_local_catalog=true
```

Mixed mode

In this mode, only some coordinators are enabled to use the metadata on-demand.

We recommend that you use the mixed mode only for testing local catalog's impact on heap usage.

Set the following on catalogd:

```
--catalog_topic_mode=mixed
```

Set the following on impalad coordinators with metdadata on-demand:

```
--use_local_catalog=true
```

Flags related to use_local_catalog

When use_local_catalog is enabled or set to True on the impalad coordinators the following list of flags configure various parameters as described below. Cloudera recommends maintaining the

default values for the flags; however, you can modify the MAX_STMT_METADATA_LOADER _THREADS flag to suit your environment.

- The flag local_catalog_cache_mb (defaults to -1) configures the size of the catalog cache within each coordinator. With the default set to -1, the cache is auto-configured to 60% of the configured Java heap size. Note that the Java heap size is distinct from and typically smaller than the overall Impala memory limit.
- The flag local_catalog_cache_expiration_s (defaults to 3600) configures the expiration time of the catalog cache within each impalad. Even if the configured cache capacity has not been reached, items are removed from the cache if they have not been accessed in the defined amount of time.
- The flag local_catalog_max_fetch_retries (defaults to 40) configures the maximum number of retries needed for queries to fetch a metadata object from the impalad coordinator's local catalog cache.
- The MAX_STMT_METADATA_LOADER_THREADS flag (defaults to 8) controls the maximum number of threads used to load table metadata during query compilation. In local catalog mode, you can parallelize this process to reduce round-trip latency to the Catalog service, which improves performance for complex queries involving many tables. If a query requires metadata for only one table, or if you set this flag to 1, Impala loads the metadata sequentially.

  The following example shows how to increase the parallelism for a query that joins many tables:

  ```
  SET MAX_STMT_METADATA_LOADER_THREADS=16;
  SELECT * FROM table1 JOIN table2 JOIN table3...
  ```

  The following example shows how to disable parallel metadata loading by setting the thread count to 1:

  ```
  SET MAX_STMT_METADATA_LOADER_THREADS=1;
  SELECT count(*) FROM functional.alltypes;
  ```

Limitation:

HDFS caching is not supported in On-demand metadata mode coordinators.

Note:

In Impala 3.4.0 and above, global INVALIDATE METADATA statement is supported when On-demand feature is enabled.

INVALIDATE METADATA Usage Notes:

To return accurate query results, Impala needs to keep the metadata current for the databases and tables queried. Through "automatic invalidation" or "HMS event polling" support, Impala automatically picks up most changes in metadata from the underlying systems. However there are some scenarios where you might need to run INVALIDA TE METADATA or REFRESH.

- if some other entity modifies information used by Impala in the metastore, the information cached by Impala must be updated via INVALIDATE METADATA or REFRESH,
- if you have "local catalog" enabled without "HMS event polling" and need to pick up metadata changes that were done outside of Impala in Hive and other Hive client, such as SparkSQL,
- and so on.
- **Note:** As this is a very expensive operation compared to the incremental metadata update done by the REFRESH statement, when possible, prefer REFRESH rather than INVALIDATE METADATA.

Related Information

Invalidate Metadata Statement

Automatic Invalidation of Metadata Cache

To keep the size of metadata bounded, the Impala Catalog Server periodically scans all the tables and invalidates those not recently used.

There are two types of configurations in Catalog Server that control the automatic invalidation of metadata in the Catalog Server Command Line Argument Advanced Configuration Snippet (Safety Valve) field in Cloudera Manager.

Time-based cache invalidation

Catalogd invalidates tables that are not recently used in the specified time period (in seconds).

The ##invalidate_tables_timeout_s flag needs to be applied to both impalad and catalogd.

Memory-based cache invalidation

When the memory pressure reaches 60% of JVM heap size after a Java garbage collection in cata logd, Impala invalidates 10% of the least recently used tables.

The ##invalidate_tables_on_memory_pressure flag needs to be applied to both impalad and cata logd.

Automatic invalidation of metadata provides more stability with lower chances of running out of memory, but the feature could potentially cause performance issues and may require tuning.

Automatic Invalidation/Refresh of Metadata

In this release, you can invalidate or refresh metadata automatically after changes to databases, tables or partitions render metadata stale. You control the synching of tables or database metadata by basing the process on events. You learn how to access metrics and state information about the event processor.

When tools such as Hive and Spark are used to process the raw data ingested into Hive tables, new HMS metadata (database, tables, partitions) and filesystem metadata (new files in existing partitions/tables) are generated. In previous versions of Impala, in order to pick up this new information, Impala users needed to manually issue an INVALIDATE or REFRESH commands.

When automatic invalidate/refresh of metadata is enabled,, the Catalog Server polls Hive Metastore (HMS) notification events at a configurable interval and automatically applies the changes to Impala catalog.

Impala Catalog Server polls and processes the following changes.

- Refreshes the tables when it receives the ALTER TABLE event.
- Refreshes the partition when it receives the ALTER, ADD, or DROP partitions.
- Adds the tables or databases when it receives the CREATE TABLE or CREATE DATABASE events.
- Removes the tables from catalogd when it receives the DROP TABLE or DROP DATABASE events.
- Refreshes the table and partitions when it receives the INSERT events.

  If the table is not loaded at the time of processing the INSERT event, the event processor does not need to refresh the table and skips it.
- Changes the database and updates catalogd when it receives the ALTER DATABASE events. The following changes are supported. This event does not invalidate the tables in the database.
  - Change the database properties
  - Change the comment on the database
  - Change the owner of the database
  - Change the default location of the database

    Changing the default location of the database does not move the tables of that database to the new location. Only the new tables which are created subsequently use the default location of the database in case it is not provided in the create table statement.

This feature is controlled by the ##hms_event_polling_interval_s flag. Start the catalogd with the ##hms_event_polling_interval_s flag set to a positive integer to enable the feature and set the polling frequency in seconds. We recommend the value to be less than 5 seconds.

Limitations

The following use cases are not supported:

- When you bypass HMS and add or remove data into table by adding files directly on the filesystem, HMS does not generate the INSERT event, and the event processor will not invalidate the corresponding table or refresh the corresponding partition.

  It is recommended that you use the LOAD DATA command to do the data load in such cases, so that event processor can act on the events generated by the LOAD command.
- The Spark API that saves data to a specified location does not generate events in HMS, thus is not supported. For example:

  ```
  Seq((1, 2)).toDF("i", "j").write.save("/user/hive/warehouse/spark_etl.db/customers/date=01012019")
  ```
- Event processing could have delays due to the polling interval and auto-refresh on large tables also takes time. If you want the metadata to be synced up immediately, manual REFRESH/INVALIDATE is a better choice and has a better guarantee.

Disable Event Based Automatic Metadata Sync

When the ##hms_event_polling_interval_s flag is set to a non-zero value for your catalogd, the event-based automatic invalidation is enabled for all databases and tables. If you wish to have the fine-grained control on which tables or databases need to be synced using events, you can use the impala.disableHmsSync property to disable the event processing at the table or database level.

This feature can be turned off by setting the ##hms_event_polling_interval_s flag set to 0.

The disable_hms_sync_by_default option controls the default event synchronization behavior for the catalog service:

- If set to true, the catalog service skips HMS event processing for all tables and databases by default. You can opt-in specific tables or databases by setting their impala.disableHmsSync property to false.
- If set to false, the catalog service processes events as usual unless specifically disabled at the table or database level.

Note: HMS notification events polling remains independent and is not affected by this flag.

When you add the DBPROPERTIES or TBLPROPERTIES with the impala.disableHmsSync key, the HMS event based sync is turned on or off. The value of the impala.disableHmsSync property determines if the event processing needs to be disabled for a particular table or database.

- If 'impala.disableHmsSync'='true', the events for that table or database are ignored and not synced with HMS.
- If 'impala.disableHmsSync'='false' or if impala.disableHmsSync is not set, the automatic sync with HMS is enabled if the ##hms_event_polling_interval_s global flag is set to non-zero.
- To disable the event based HMS sync for a new database, set the impala.disableHmsSync database properties in Hive as currently, Impala does not support setting database properties:

  ```
  CREATE DATABASE <name> WITH DBPROPERTIES ('impala.disableHmsSync'='true');
  ```
- To enable or disable the event based HMS sync for a table:

  ```
  CREATE TABLE <name> ... TBLPROPERTIES ('impala.disableHmsSync'='true' | 'false');
  ```

- To change the event based HMS sync at the table level:

```
ALTER TABLE <name> SET TBLPROPERTIES ('impala.disableHmsSync'='true' | '
false');
```

When both table and database level properties are set, the table level property takes precedence. If the table level property is not set, then the database level property is used to evaluate if the event needs to be processed or not.

If the property is changed from true (meaning events are skipped) to false (meaning events are not skipped), you need to issue a manual INVALIDATE METADATA command to reset event processor because it doesn't know how many events have been skipped in the past and cannot know if the object in the event is the latest. In such a case, the status of the event processor changes to NEEDS_INVALIDATE.

Metrics for Event Based Automatic Metadata Sync

You can use the web UI of the catalogd to check the state of the automatic invalidate event processor.

By default, the debug web UI of catalogd is at http://*IMPALA-SERVER-HOSTNAME*:25020 (non-secure cluster) or https://*IMPALA-SERVER-HOSTNAME*:25020 (secure cluster).

Under the web UI, there are two pages that presents the metrics for HMS event processor that is responsible for the event based automatic metadata sync.

- /metrics#events
- /events

  This provides a detailed view of the metrics of the event processor, including min, max, mean, median, of the durations and rate metrics for all the counters listed on the /metrics#events page.

The /metrics#events page provides the following metrics about the HMS event processor.

Name	Description
events-processor.avg-events-fetch-duration	Average duration to fetch a batch of events and process it.
events-processor.avg-events-process-duration	Average time taken to process a batch of events received from the Metastore.
events-processor.events-received	Total number of the Metastore events received.
events-processor.events-received-15min-rate	Exponentially weighted moving average (EWMA) of number of events received in last 15 min. This rate of events can be used to determine if there are spikes in event processor activity during certain hours of the day.
events-processor.events-received-1min-rate	Exponentially weighted moving average (EWMA) of number of events received in last 1 min. This rate of events can be used to determine if there are spikes in event processor activity during certain hours of the day.
events-processor.events-received-5min-rate	Exponentially weighted moving average (EWMA) of number of events received in last 5 min. This rate of events can be used to determine if there are spikes in event processor activity during certain hours of the day.
events-processor.events-skipped	Total number of the Metastore events skipped. Events can be skipped based on certain flags are table and database level. You can use this metric to make decisions, such as: • If most of the events are being skipped, see if you might just turn off the event processing. • If most of the events are not skipped, see if you need to add flags on certain databases.

Name	Description
events-processor.status	Metastore event processor status to see if there are events being received or not. Possible states are: • PAUSED The event processor is paused because catalog is being reset concurrently. • ACTIVE The event processor is scheduled at a given frequency. • ERROR • The event processor is in error state and event processing has stopped. • NEEDS_INVALIDATE The event processor could not resolve certain events and needs a manual INVALIDATE command to reset the state. • STOPPED The event processing has been shutdown. No events will be processed. • DISABLED The event processor is not configured to run.

Hierarchical metastore event processing

Hierarchical metastore event processor uses a multi-layered approach to improve synchronization speed and handle event dependencies.

In Cloudera Runtime 7.3.2 and higher versions, the metastore event processor supports a multi-layered approach to improve synchronization speed and handle event dependencies.

By default, the metastore event processor is single-threaded. Notification events are processed sequentially with a maximum limit of 1000 events fetched and processed in a single batch. Multiple locks address concurrency issues that can arise when catalog DDL operation processing and metastore event processing try to access or update catalog objects at the same time.

Waiting for a lock or file metadata loading of a table can slow event processing and affect subsequent events, even if those events are not dependent on the previous one. This results in a long synchronization time for Hive Metastore (HMS) events.

Multi-level event processing

You can convert existing metastore event processing into multi-level event processing by using the enable_hierarchical_event_processing flag. This method partitions events based on their dependency, maintains the order of events within that dependency, and processes them independently when possible.

The hierarchical model consists of the following layers:

- **Event dispatcher** – Dispatches metastore events and maintains a fixed pool of database event executors.
- **Database event executor** – Manages table event dispatching, processes database events, and manages table event executors.
- **Table event executor** – Processes events for multiple tables by using table processors.

Multi-level event processing ensures linearizability for a specific table by processing all table-specific events in the order they occurred.

Event synchronization and ordering

To maintain metadata consistency, certain events act as synchronization barriers and are not processed in parallel:

- A **database synchronization barrier** restricts the table processors of the database from processing events that occur after the database event, until the database event is fully processed.

- A **rename synchronization barrier** wraps an `ALTER TABLE` rename event. It synchronizes the source and target table processors to ensure the source processor removes the table before the target processor creates the renamed table.
- A transaction synchronization barrier is used when the system creates pseudo-commit and pseudo-abort transaction events for each table involved, because `COMMIT_TXN` and `ABORT_TXN` can involve multiple tables. These pseudo events are then processed independently at their respective table processors.

Catalogd configuration properties

Use the following properties to configure the event executors and processing limits for the metastore event processor.

Table 1: Event Processor Configuration Properties

Property	Description
enable_hierarchical_event_processing	Enables hierarchical event processing on Catalogd. This is disabled by default.
num_db_event_executors	Sets the number of database-level event executors.
num_table_event_executors_per_db_event_executor	Sets the number of table-level event executors within a database event executor.
min_event_processor_idle_ms	Sets the minimum time to retain idle database and table processors when they have no events to process.
max_outstanding_events_on_executors	Sets the maximum number of outstanding events to process on event executors.
hms_event_polling_interval_s	Defines the interval for polling events. This property supports double values to allow for millisecond precision.

OpenTelemetry support for Impala

The Impala OpenTelemetry integration enables real-time query observability and centralized telemetry data collection, including lifecycle events and resource usage.

Overview

OpenTelemetry (OTel) provides an open-source solution for collecting, processing, and exporting telemetry data, including metrics from applications. OTel helps users gain visibility into query performance and troubleshoot query failures. Impala supports OpenTelemetry (OTel) in Cloudera Runtime 7.3.2.

Impala telemetry data is integrated with OTel-compatible collectors. This provides a centralized flow of live query insights, with SELECT queries, DMLs, and DDLs represented as OTel traces, and reduces the friction of sourcing data from multiple places.

Impala integration with OTel

Impala integrates the OTel C++ SDK to emit query lifecycle data as OpenTelemetry traces. The system already tracks specific phases and events for each query and records them in the query profile timeline section. By emitting these events to an OTel collector, observability systems can track active queries in near real-time.

Collected telemetry data

Telemetry data emitted from Impala carries crucial information that is currently available only in the query profile and workload management tables. Telemetry data includes the following data:

1. The initiating user
2. The SQL statement
3. Memory estimates and actual use

4. Other important data related to the query lifecycle

Related Information

OpenTelemetry

Benefits of OTel Impala Integration

Learn about the benefits of integrating OpenTelemetry (OTel) with Impala focusing on enhanced observability, telemetry data insights, and optimized performance.

Enhanced query lifecycle and historical data

Integrating OTel with Impala provides enhanced observability through comprehensive data collection. The following data is collected:

- Live query data – Data about important events in the lifecycle of actively running queries is sent to collectors in near real-time as the events happen.
- Historical query data – Data about completed queries can be retained by the destination OTel trace management system.

Performance and scalability impact

The integration is designed to have negligible impact on Impala's performance and is built for scalability. The following performance and scalability impacts are valid for the integration:

- The performance impact on Impala is negligible because the system already collects all the necessary event and metric data.
- The process of sending data to the OTel endpoint is handled out-of-band in a separate thread, limiting any additional resource usage to just the sending of this data.
- Scalability concerns are primarily limited to the OTel Collector endpoint. Impala does not encounter scalability issues as long as communication with the Collector happens without delay.

Configuring OTel in Impala

Learn how you can enable OTel in Impala and configure certain properties that enable you to optimize the data collection.

Before you begin

- Ensure that you are on Cloudera Runtime 7.3.2 or higher version.
- To ensure secure and reliable telemetry data transfer, you can configure SSL or TLS, and retry settings. These flags apply specifically to the communication between the Impala daemon and the OTel collector.

Procedure

1. In Cloudera Manager, click Clusters Impala Configuration .
2. Search for **Impala Daemon Command Line Argument Advanced Configuration Snippet (Safety Valve)**
3. Manually add the configurations as needed:
   a) Add the --otel_trace_collector_url configuration key.

      Specifies the URL of the OpenTelemetry collector to which trace data will be exported. The endpoint must be a valid URL.

      ```
      --otel_collector_url= https://collector-endpoint
      ```

Note: For secure data transmission, Cloudera recommends using https in the URL.

b) Add the --otel_trace_enabled configuration key. The default value is false.
   If set to true, OpenTelemetry traces will be generated and exported to the configured OpenTelemetry collector.
c) Add the otel_trace_batch_max_batch_size configuration key. The default value is 512.
   Specifies the maximum batch size of every export to the OTel Collector.
d) Add the otel_trace_batch_queue_size configuration key. The default value is 2048.
   Specifies the maximum buffer or queue size. After the set size is reached, spans are dropped.
e) Add the otel_trace_batch_schedule_delay_ms configuration key. The default value is 5000.
   Specifies the delay interval in milliseconds between two consecutive batch exports.
f) Add the otel_trace_additional_headers configuration key.
   Specifies a list of additional HTTP headers to be sent with each call to the OTel Collector .
g) Add the otel_trace_ca_cert_path configuration key.
   Specifies the path to a file containing a CA certificates bundle.
h) Add the otel_trace_ca_cert_string configuration key.
   Specifies a string containing a CA certificates bundle.
i) Add the otel_trace_compression configuration key. The default value is true.
   If set to true, uses ZLib compression for sending data to the OTel Collector.
j) Add the otel_trace_timeout_s configuration key. The default value is 10.
   Specifies the export timeout in seconds.
k) Add the otel_trace_tls_insecure_skip_verify configuration key. The default value is false.
   If set to true, skips verification of the collector TLS certificate.

Note: Cloudera recommends setting this configuration to true only for development or testing purposes.

l) Add the otel_trace_tls_minimum_version configuration key.
   The default value is the overall minimum TLS version from the ssl_minimum_version flag.
m) Add the ssl_minimum_version configuration key. The default value is tlsv1.2.
   Specifies the minimum SSL or TLS version that Impala Thrift services are expected to use for both client and server connections. This flag applies to all Impala Thrift services, and supported versions include TLSv1, TLSv1.1, and TLSv1.2.

Important: The value of this flag is used if the otel_trace_tls_minimum_version configuration key is not specified.

n) Add the otel_trace_ssl_ciphers configuration key. The default value is the value of Impala ssl_cipher_list startup flag.
   Specifies a list of allowed TLS cipher suites when using TLS 1.2.
o) Add the otel_trace_tls_cipher_suites configuration key. The default value is the value of Impala tls_ciphersuites startup flag.
   Specifies a list of allowed TLS cipher suites when using TLS 1.3.
p) Add the otel_trace_retry_policy_max_attempts configuration key. The default value is 5.
   Specifies the maximum number of call attempts, including the original attempt.
q) Add the otel_trace_retry_policy_initial_backoff_s configuration key. The default value is 1.
   Specifies the initial backoff delay between retry attempts in seconds.
r) Add the otel_trace_retry_policy_max_backoff_s configuration key. The default value is 0.
   Specifies the maximum backoff delay between retry attempts in seconds. A value of 0 or less indicates that the configuration key is not set.
s) Add the otel_trace_retry_policy_backoff_multiplier configuration key. The default value is 2.

Specifies the factor by which the retry interval increases after every failed attempt.

t) Add the otel_debug configuration key. The default value is false.

If set to true, this outputs additional debug information.

4. Save the changes and restart the Impala service

Telemetry data exposed to OTel collector for Impala

OpenTelemetry (OTel) provides an open-source solution for collecting, processing, and exporting telemetry data. In Impala, query lifecycle data is structured as OTel traces, with a root span and multiple child spans. The following tables detail the attributes and events associated with these traces.

Root Span of the Trace

The root span represents the entire query and is of kind SERVER.

Metrics	Type	Description	Example
ClusterId	String	A string that uniquely identifies a cluster, determined by the value of the --cluster_id startup flag.	impala-1982661901-6j1z
EndTime	Millisecond epoch time	The time when the query finished in millisecond epoch time.	1743474118301
ErrorMessage	String	A string containing the error message received from the query execution, or an empty string if the query completed successfully.	Invalid syntax
OriginalQueryId	String	When a query is retried, a string containing the Impala query ID of the original query. Otherwise, an empty string.	ef403b2690d243be:b960c0ba00000000
QueryId	String	The Impala query ID.	ef403b2690d243be:b960c0ba00000000
QueryStartTime	Millisecond epoch time	The time when the query was first received in millisecond epoch time.	1743473803667
RequestPool	String	A string containing the name of the request pool the query will be scheduled into.	default-pool
RetriedQueryId	String	The ID of the query that successfully retried this query.	3a43a57ad0bf36df:dcd698b700000000
Runtime	Milliseconds	The time in milliseconds the query ran.	5231
SessionId	String	The Impala session ID.	024f45f3e0019fed:eb0fe00c00000000
State	String	The query state.	FINISHED, EXCEPTION, or RETRIED
QueryType	String	The type of the statement in uppercase letters, such as QUERY, DML, or DDL.	DML
UserName	String	The user who submitted the query.	usr123

Child spans

Child spans are of kind INTERNAL and contain both global and specific attributes and events.

Global child span attributes

The global child span attributes are present on every child span.

Metrics	Type	Description	Example
BeginTime	Millisecond epoch time	The time the span started in millisecond epoch time.	1743473803667
ElapsedTime	Millisecond	The time in milliseconds the span ran.	5231
EndTime	Millisecond epoch time	The time the span finished in millisecond epoch time.	1743474118301
ErrorMsg	String	Error details if a failure occurred during this span.	Could not read file
Name	String	The name of the span, in the {{query_id}} - Query Stage format.	cdf601fa776431c43:5e59cba4fe284ae22 - Submitted
Running	Boolean	A boolean value indicating if the query is actively running and not in planning, admission control, or closing. It is set to false if the query fails during this span.	true
Status	String	The status of the task, for example, OK.	OK

Specific child span attributes

The following table lists the additional attributes for the Init span, which are unique to that stage of the query lifecycle.

Metrics	Type	Description	Example
ClusterId	String	A string that uniquely identifies a cluster, determined by the value of the --cluster_id startup flag.	impala-1982661901-6j1z
DefaultDb	String	The name of the default database.	tpcds
Name	String	The name of the span, in the {{query_id}} - Init format.	cdf601fa776431c43:5e59cba4fe284ae22 - Init
OriginalQueryId	String	The Impala query ID of the original query when retried. Otherwise, an empty string.	ef403b2690d243be:b960c0ba00000000
QueryId	String	The Impala query ID.	ef403b2690d243be:b960c0ba00000000
QueryString	String	The redacted string containing the SQL statement.	select * from db.tbl where col1 = “val1”
RequestPool	String	A string containing the name of the request pool the query will be scheduled into.	default-pool
SessionId	String	The Impala session ID.	024f45f3e0019fed:eb0fe00c00000000

Metrics	Type	Description	Example
UserName	String	The user who submitted the query.	usr123

Query Submitted

Metrics	Type	Description	Example
Name	String	The name of the span, in the {{query_id}} - Init format.	cdf601fa776431c43:5e59cba4fe284ae22 - Submitted

Span: Planning

Metrics	Type	Description	Example
Name	String	The name of the span, in the {{query_id}} - Init format.	cdf601fa776431c43:5e59cba4fe284ae22 - Planning
QueryType	String	The type of the statement in uppercase letters, such as QUERY, DML, or DDL.	DML, QUERY

Admission control

Note: The admission control attributes are only applicable for QUERY and DML statements.

Metrics	Type	Description	Example
Name	String	The name of the span, in the {{query_id}} - AdmissionControl format.	cdf601fa776431c43:5e59cba4fe284ae22 - AdmissionControl
AdmissionResult	String	A string containing the result from admission control.	Admitted immediately
Queued	Boolean	A boolean value where true indicates that the query was queued and false indicates that the query was admitted immediately.	false
RequestPool	String	A string containing the name of the request pool the query will be scheduled into.	default-pool

Query execution

Metrics	Type	Description	Example
Name	String	The name of the span, in the {{query_id}} - Execution format.	cdf601fa776431c43:5e59cba4fe284ae22 - Execution
NumDeletedRows	Integer	An integer containing the total number of rows deleted by the DML statement.	0
NumModifiedRows	Integer	An integer containing the total number of rows modified by the DML statement.	0
NumRowsFetched	Integer	An integer containing the total number of rows fetched by the client.	5132

Close

Metrics	Type	Description	Example
Name	String	The name of the span, in the {{query_id}} - Close format.	cdf601fa776431c43:5e59cba4fe284ae22 - Close

OpenTelemetry Impala query tracing example

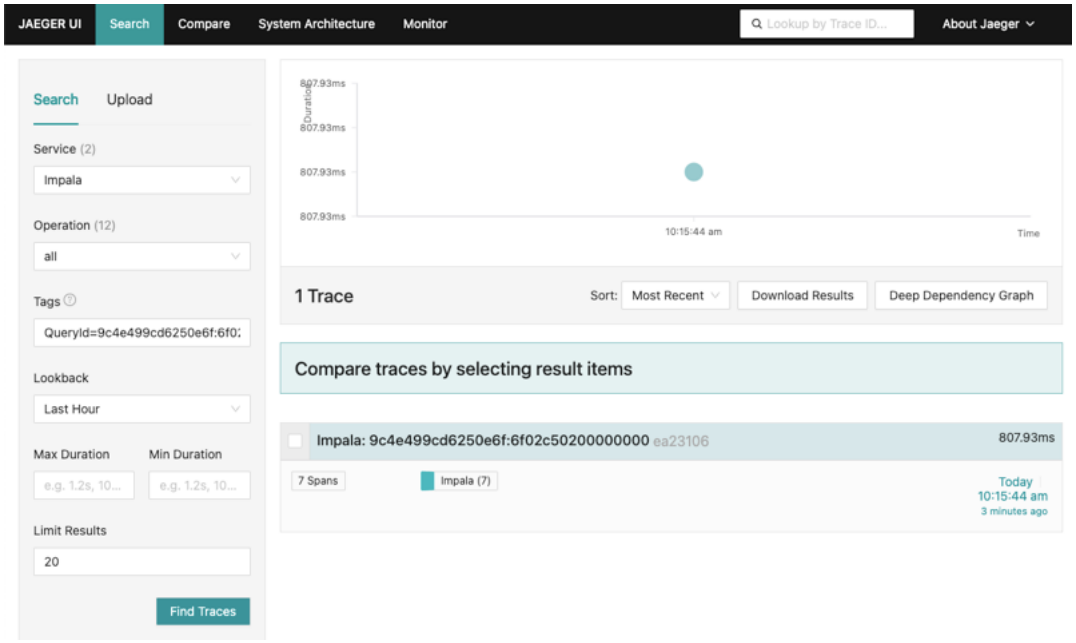
Learn how to use Jaeger to visualize the telemetry data from a simple Impala query.

Important: The example backend systems shown here are just for representation. Along with having your own instance of the OTel collector, you must also have your own instances of Jaeger, Zipkin, Prometheus and Grafana to view the telemetry data.

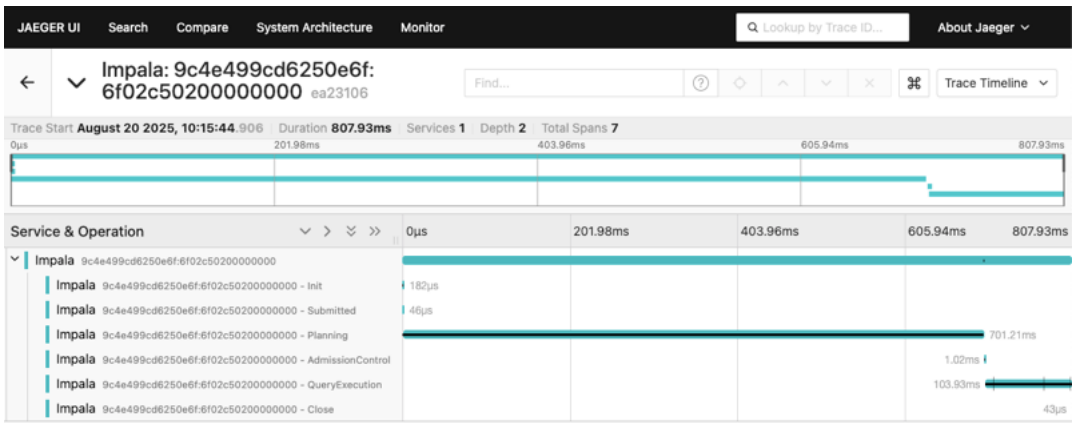
After configuring OpenTelemetry support for your Impala Virtual Warehouse, you can view detailed telemetry data for your queries in a trace visualization system like Jaeger. The following steps show a typical workflow for finding and analyzing a query trace.

Finding a query trace

1. On the Jaeger UI, use the search function to find traces. You can filter by service, such as Impalad, or other tags like the query ID. The search results provide a high-level summary of the traces found.



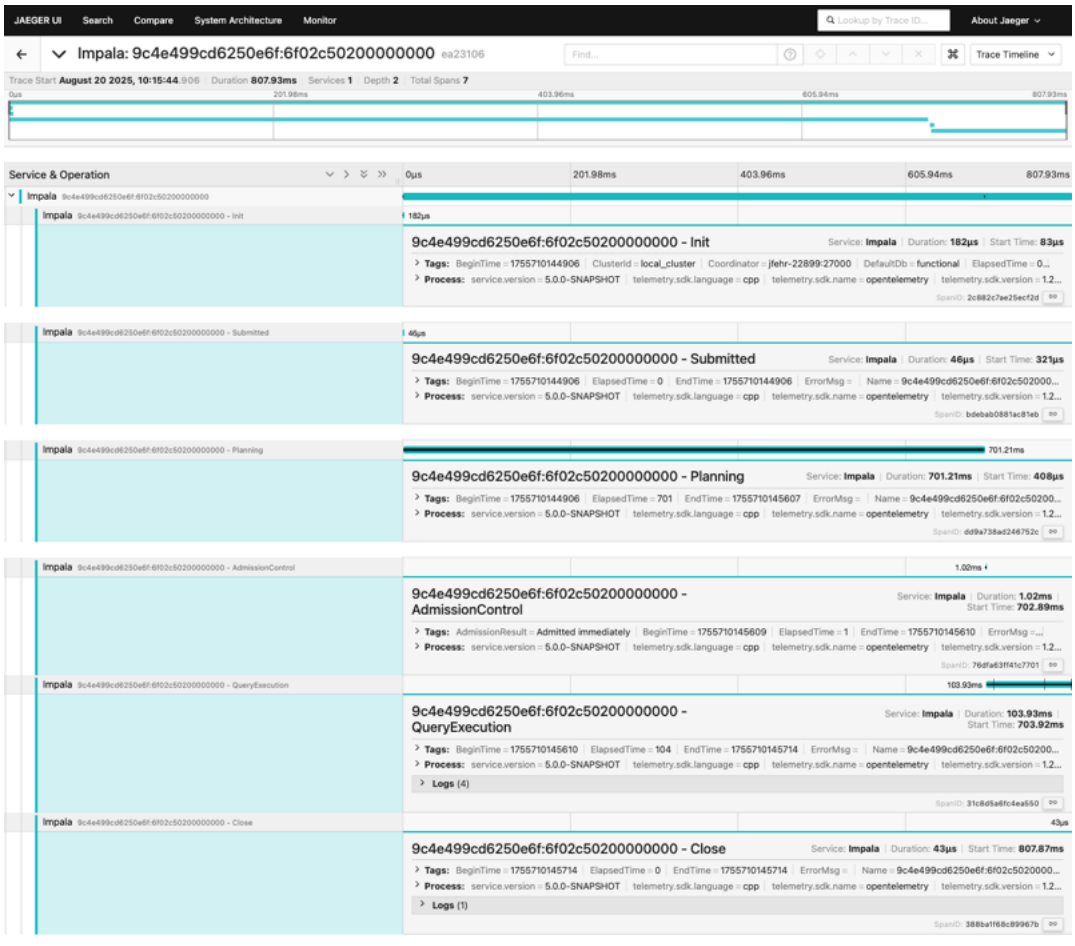
2. From the search results, select a specific trace to view its summary and timeline.



Understanding the query trace timeline

A trace provides a detailed breakdown of a query execution from start to finish. Each segment in the timeline is a span, representing a specific operation. The main query is represented by a root span, and its various stages are shown as child spans.

The following image shows the full timeline of a simple query, with a breakdown of its core stages.

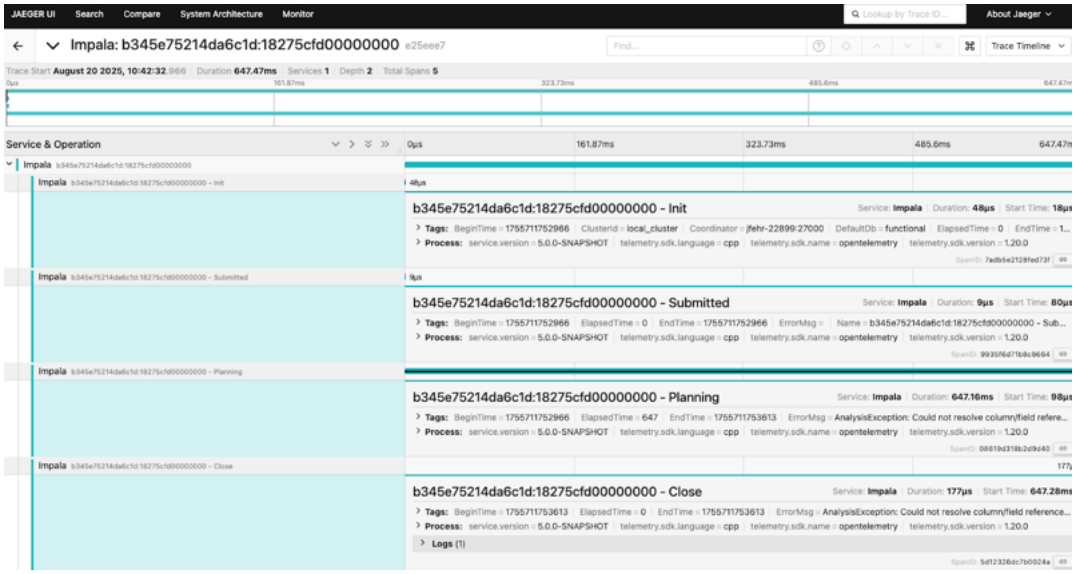


Analyzing child spans

By inspecting the individual child spans, you can gain deep insights into the performance of each stage of the query. For example, you can see how much time was spent on planning, admission control, and execution.

The following query stages are shown in the child spans:

1. Query initialization – This span is generated immediately when a query is received to capture the initial setup and is useful for checking early details, such as the user and query ID.
2. Query planning – This span shows the time taken to plan the query. A long duration here might indicate a complex query or metadata-related issues.
3. Admission control – This span records the time spent waiting for resources. A long duration here suggests high resource contention.
4. Query execution – This span represents the time for the actual query to run. This is a crucial metric for evaluating performance.
5. Query close – This span marks the end of the query lifecycle, including cleanup and final reporting.
6. Failed query – A failed query will also generate a trace that can be analyzed to understand the cause of the failure. In this example, the query failed during the planning phase, likely due to a syntax error or a non-existent table. The root span is marked with an error tag, and the logs provide details on the failure.



Analyzing these spans helps you identify performance bottlenecks and understand the entire lifecycle of a query within your Impala Virtual Warehouse.

Limitation of OpenTelemetry support for Impala

Learn about the current limitations of OpenTelemetry support in Impala, including restricted scope and lack of customization.

The current OpenTelemetry integration for Impala has the following limitations:

Telemetry data scope

The scope of telemetry data is currently limited to only traces of select queries, DMLs, and DDLs. Other types of data such as metrics or logs will not be handled in the initial release. However, the design is prepared to allow metrics or logs to be added in the future without significant architectural changes.

Select queries that leverage Common Table Expressions will not have traces generated for them.

Fixed trace data

The traces being sent from Impala are not configurable. This means you cannot customize the specific data points included in the traces.

Synchronization between Impala Clusters

Describes the event that notifies other Impala clusters about the changes of LOAD data statement.

When a LOAD statement is run from an Impala in a cluster, an INSERT event is generated. This INSERT event generated notifies other Impala clusters or any other systems that consume HMS events (e.g. Hive Replication) about the changes of LOAD DATA. The other Impala clusters will refresh the table or partitions based on the event. This allows the metadata to be always consistent locally and remotely. However, there can be delays till the event is processed but eventually, it will be consistent. Also, an auto-refresh on large tables takes time. If you want the metadata to be synced up immediately, manual REFRESH/INVALIDATE is a better choice and has a better guarantee.

When the LOAD DATA statement operates on a partitioned table, it always operates on one partition at a time. So for unpartitioned tables, the granularity of the refresh is the whole table. For partitioned tables, only the partition will be reloaded.