Administering Jethro

Changing the Password of Jethro User

Each Jethro instance has a predefined database user called **jethro**, with a default password jethro. To change the default password, run the following command from any Jethro host:

```
JethroAdmin change-pass instance_name
```

You will be asked to provide a new password for the **jethro** user.
The current release uses this user/password combination to control access to the instance. A more comprehensive authentication and authorization functionality will be available in a future release.

Reviewing the Local Directory Structure

Jethro software is installed under **/opt/jethro**.
The directory structure supports multiple installed binary versions - for example, for easier upgrade / downgrade. It also supports multiple instances - for example, running several dev and test instances from the same host.
/opt/jethro - The top Jethro directory
/opt/jethro/jethro-version - One directory per installed version of Jethro
/opt/jethro/jethro-version/bin - Binaries and scripts
/opt/jethro/jethro-version/conf - Templates for new instance configuration
/opt/jethro/jethro-version/doc - Documentation and samples
/opt/jethro/jethro/jethro-version/instances - Symbolic link to *instances* directory
opt/jethro/jethro/jethro-version/jdbc - JDBC driver
/opt/jethro/jethro/jethro-version//lib - Shared libraries
/opt/jethro/current - A symbolic link to the current version of Jethro
/opt/jethro/instances - Metadata for all instances
/opt/jethro/instances/services.ini - A config file controlling autostart of services
/opt/jethro/instances/instance_name - Metadata of a specific instance
/opt/jethro/instances/instance_name/local-conf.ini - A config file with the instance HDFS location, local cache attributes and overriding parameters
/opt/jethro/instances/instance_name/local-cache.ini - A config file for additional control of local caching (see section Managing Local Cache)
/opt/jethro/instances/instance_name/jethrolog.properties- A config file controlling instance debug logging (used by Jethro support)
/opt/jethro/instances/instance_name/log - Symbolic link to the instance log dir
/var/log/jethro - Top directory for Jethro logs
/var/log/jethro/instance_name - Log directory for a specific instance

Managing Local Cache

JethroServer automatically caches some of the JethroData model files (columns and indexes). Caching can accelerate read time for frequently accessed files, as well as avoiding frequent accesses to the NameNode and DataNodes.The local cache is populated by the **JethroServer** process in the background. It is used by all Jethro processes, including **JethroLoader** and **JethroMaint**.

The local cache is populated by the **JethroServer** process in the background. It is used by all Jethro processes, including **JethroLoader** and **JethroMaint**.
The cache transparently handles partially written files – if a file is cached and later appended with new data, all processes will continue to work properly (internally, the older data will be read from the local cache and the newer parts will be read directly from HDFS).
In addition, the local cache is automatically checked and updated once a minute. Irrelevant files are removed (after tables are dropped or indexes are optimized) and new data is incrementally added, up to the cache maximum size limit.

The local cache location and size is controlled by the parameters **local.cache.path** and **local.cache.max.size**. Both parameters are initially derived from the command line parameters of **JethroAdmin create-instance** or **attach-instance**, which provides a directory that is used for both Local Cache and Adaptive Cache. They can be modified later, if needed, by editing the */local-conf.ini* file at **$JETHRO_HOME/instances/{instance_name}.**

Full Caching of Specific Columns

In addition to the automatic caching, you can explicitly specify that specific columns or tables will be fully cached; for example, frequently accessed columns of large tables (such as join keys or common measures), or an entire small dimension table.
In the current release, you need to manually specify which columns or tables are fully cached, by using the configuration file */local-cache.ini* in **$JETHRO_HOME/instances/instance_name**.
Each line in the file adds a caching request for a column or a table in the following format:
schema_name.table_name.column_name

To cache all columns, specify * instead of column name,. For example:
def_schema.country_dim.*
def_schema.time_dim.*
def_schema.fact.country_id
def_schema.fact.time_id

Processing Order – The local cache is first populated with schema and column metadata. Then, the */local-cache.ini* file is processed line by line, so the order of lines in this file implies priority. In any case, local caching will not exceed the user-specified maximum disk space, defined by the parameter **local.cache.max.size**).

Refreshing the file – Changes to */local-cache.ini* are automatically identified and applied by the JethroServer process during the next cache refresh (once a minute). There is no need for any user action such as restarting the process.

Errors – if any errors are encountered while parsing the file, they will be noted in the JethroServer log file at ***$JETHRO_HOME/instances/instance_name/log/jethroserver.log.***

Viewing the Local Cache

The current contents of the local cache can be displayed by running the following command: ***SHOW LOCAL CACHE***
The output of this command displays the disk space each table uses in the local cache. In addition, under the *Non-metadata cached (MB)* column you can find how much of the total cache comes from the specific */local- cache.ini* configuration.

```
SHOW LOCAL CACHE;
Schema Name | Table Name | Total Cached (MB)| Non-metadata cached (MB) -
 def_schema | test1 | 1,713.31 | 1,008.73 def_schema | test2 | 6,035 | 0 def_schema| | test3 | 20,800.9 |
18,634 TOTAL | TOTAL | 22,520.245 | 19,642.73
-------------------------------------------------------------------------
```

If you modify the local cache configuration, the **SHOW LOCAL CACHE** command can help you monitor the progress of populating the cache and the size cached per table

Managing Adaptive Cache

As users work with their favorite BI tools, the sequence of SQL queries that these tools generate and send to Jethro has some predictable patterns. For example, most users start from a dashboard (that sends a predictable list of queries), than typically start adding filter conditions and aggregate conditions one at a time.
Adaptive cache is a unique JethroData feature, which leverages those patterns. It is a cache of re-usable, intermediate bitmaps and query results that is automatically maintained and used by the **JethroServer** engine on its local storage.

The cache is also **incremental** – after new rows are loaded, the next time a cached query is executed it will in most cases automatically combine previously cached results, and computation will only be performed over the newly loaded rows.

The adaptive cache is managed locally by each **JethroServer**. It is guaranteed to be consistent – having the same transactional visibility as a regular query – and does not return stale results. The adaptive cache is self-managed – when it fills up, it removes less useful entries while taking into account how frequently they were used, how recently they were used, and how much cache space they consume.
This section contains the following components:

Cache Size and Location

The adaptive cache location and size is controlled by the parameters **adaptive.cache.location** and **adaptive.cache.quota**. Both parameters are initially derived from the command line parameters of **JethroAdmin create-instance** or **attach-instance**, which provides a directory that is used for both Local Cache and Adaptive Cache. They can be modified later, if needed, by editing the */local-conf.ini* file at **$JETHRO_HOME/instances/{instance_name}**.

Adaptive Query Cache

Adaptive query cache is the part of the adaptive cache that holds **intermediate query result sets** (for SELECT statements only). It is enabled by default, but can optionally be turned off by setting **adaptive.cache.query.enable** to zero (this could be useful when performing static performance benchmarks).
A query is considered for inclusion in the adaptive query cache if its running completed without errors and took more than **adaptive.cache.query.min.response.time**, which defaults to one second. Also, the cached result set must be lower than **adaptive.cache.query.max.results** rows, which defaults to 100,000 rows. When possible, the result set is stored in incremental format, otherwise it is stored in full format.

Adaptive Query Cache - Incremental
Jethro can incrementally refresh several common types of queries after new data is loaded - for example, aggregate queries on a star schema. In those cases, when a query result set was stored in the query cache and new data is loaded into one of the tables, the next time this query is executed Jethro will automatically combine the previously cached result set and perform computation only on the newly inserted rows.
A query or a sub-query must meet several criteria to be cached in incremental result set mode. The main ones are:

- It must be GROUP BY query
- The number of rows returned from the top-level GROUP BY, before the final HAVING clause or LIMIT clause, must not exceed the limit defined by the following setting: ***cache.query.max.results.***

Adaptive Query Cache - Full mode
When a query does not meet the criteria to be cached in incremental result set format, it is considered for being fully stored. Any query can have its result set fully stored if its number of rows is below the threshold, except queries that call non-deterministic functions, such as **rand**() or **now**().
In this mode, the final query result set is stored. As a result, repeated query executions will just read the cached result and send it to the client without additional processing. However, a full result set is not incremental - if any of the tables involved in the query change (new rows inserted, old partitions dropped and so on), the cached result set will be discarded.

Adaptive Index Cache
Adaptive index cache is the part of the adaptive cache that holds intermediate bitmap index entries that were computed on the fly during query execution. It is enabled by default, but can optionally be turned off by setting **adaptive.cache.index.enable** to zero (this could be useful when performing static performance benchmarks).
An intermediate bitmap is considered for inclusion in the adaptive index cache if its creation took more than **adaptive.cache.index.min.response.time**, which defaults to 0.5 second. Also, its size must not exceed the the limit defined by the **adaptive.cache.index.max.size** parameter, which defaults to 100MB.
The intermediate bitmaps are stored separately per partition. As a result, dropping a partition does not invalidate cached bitmaps from other partitions. In addition, intermediate bitmaps are currently cached only for IN conditions, either with a list of values or those generated as part of a star transformation optimization.

Viewing the Adaptive Cache
You can view all currently cached queries and indexes by running the **SHOW ADAPTIVE CACHE ACTIVE** command. The command shows various attributes of the cached elements, including the number of times they were used, the last time they were used, and their rank - low rank indicates less valuable queries and indexes, which are more likely to be deleted when cache becomes full.

Managing Query Queuing

Query queuing is a feature that limits the number of SQL queries that can run concurrently, thereby reducing Jethro's peak memory utilization. This helps avoiding out-of-memory issues under heavy usage.

Query Queuing Behavior and Parameters

- **Concurrent Queries** - The maximum number of concurrently running queries is controlled by the parameter **query.resource.max.concurrent.queries**.
  The default value is 0, which means that Jethro automatically picks a value based on the host memory size and number of cores. Alternatively, you can set this value to any positive number, to select the number of concurrently running queries.

- *The exact formula is internal and may change from version to version.*
  As of version 0.9, it is set to allow one concurrent query per 20GB of host RAM or one concurrent query per four cores - the lower of the two.

- **Additional Memory Limit -** Jethro also adapts the number of concurrent queries based on the host's current memory utilization , as an additional protection against memory spikes.
  If available host memory drops below a certain threshold, new queries will be queued, regardless of the number of queries currently running. The threshold, which defaults to 15%, is controlled by the following parameter: ***query.resource.min.memory.available.percentage.***

- **Query Queue Size** - The number of queries that can be queued at the same time is controlled by the parameter **query.resource.max.waiting.queries**, whose default value is 20 queries. When a query needs to be queued and the queue is full, it will fail immediately.
- **Queue Timeout** - If a query is queued for execution for a long period of time, it will eventually time out and an error will be returned to the client. The timeout is controlled by the parameter **query.resource.max.waiting.timeout**, which defaults to 60 seconds. This parameter can also be set at the session level.

- **Adaptive Cache Bypass** - When a query is received from the client, the **JethroServer** checks if the results for the query are already available in the adaptive query cache, in which case the query will skip the queue and will return the cached results
- **Non-Query Statements** are never queued. They are typically not resource-intensive.

Viewing the Queries Queue

You can view all queries that are in progress by running the **SHOW ACTIVE QUERIES** command. The command's output displays all queries that are in progress (running or queued), including their position, duration, the client IP address, and the query text.
For example:

```
SHOW ACTIVE QUERIES;
Host | Pos.| Query ID | Duration | Status | Client | Query -----------------------
jethro-dev1 | 1 |6@jethro-dev1 |20.5861981 | RUNNING | 62.0.78.21 | SELECT ...
jethro-dev1 | 2 |7@jethro-dev1 |18.5715730 | RUNNING | 62.0.78.195| SELECT ...
jethro-dev1 | 3 |8@jethro-dev1 |18.5382791 | RUNNING | 62.0.74.2 | SELECT ...
jethro-dev1 | 4 |9@jethro-dev1 |14.5227342 | QUEUED | 127.0.0.1 | SELECT ...
jethro-dev1 | 5 |10@jethro-dev1 |6.4988081 | QUEUED | 62.0.74.203| SELECT ...
jethro-dev1 | 6 |14@jethro-dev1 |0.0000522 | RUNNING | 62.0.78.62 | SHOW ACTIVE...
--------------------------------------------------------------------------------
```

When using Client-Side Load Balancing, each SQL command is directed to a different Jethro host.
It is advisable that you take it into account and select whether to:

- Connect to a single **JethroServer** to investigate.
- Prepare to have each execution return from a different host (check the Host column).

- Use **JethroClient**, which has special handling for this command; it automatically iterates over all the **JethroServers** in its connect string and concatenates the results.

Viewing Background Maintenance Status

The **JethroMaint** service is in charge of running maintenance tasks in the background. Its main responsibilities are:

1. **Optimizing indexes -** Performing a background **merge** of column indexes after data loads.
   The merge writes a new version of a column index, which replaces two or more column index files. New queries will automatically start using the new version of the column index.

1. **Deleting unneeded files** - After an index was optimized, the files of its old version can be deleted. However, there could be queries already in progress that access the older version of the index, so the older versions are deleted after a delay. Dropped and truncated tables are also physically deleted in the background after the same delay.

If the **JethroMaint** service was not running for a long time (for example, it was manually stopped or it ran into an unexpected issue), it could affect query performance, as each column may have non-optimized index with many small index files.

SHOW TABLES MAINT Command and Output

To verify that all maintenance operations ran successfully, use the **SHOW TABLES MAINT** command:

```
SHOW TABLES MAINT;
ID  | Start Value |Table | Merge |Non-metadata Name Status cached (MB)) ------
806 | def_Schema | test1 | FULLY MERGED (145/145 | 311.1
1208| def_Schema | test2 | FULLY MERGED (140/28 | 2177.4
1295| def_Schema | test3 | FULLY MERGED (32/32 | 0.0
1832| TOTAL | test4 | FRAGMENTED /37) | 5436.6
-------------------------------------------------------------------------
```

The **Merge Status** column shows a status indication of the merge status:

- **FULLY MERGED** means that all of the table's indexes are fully optimized and there is no additional optimization work pending.
- **MOSTLY MERGED** means that there is only a minor amount of work of index optimization pending. Query performance should not be significantly affected, though if you run a benchmark, you should probably wait a few minutes to let **JethroMaint** complete the background optimization.
- **FRAGMENTED** means that there are many index files per column, which may negatively impact query performance. This should not happen under normal circumstances and typically indicates that **JethroMaint** was not running for an extended period (see troubleshooting below).

The Merge Status also includes a more technical status - the actual number of index files and optimal number of index files (**actual / optimal**). In most cases, the optimal number of index files are one per column (and for partitioned table, one per column per partition); however, if a table (or a partition) is very large (many billions of rows), the optimal number could be larger.
The **Pending Delete** column shows how many MBs will be freed in HDFS when old index files are deleted by JethroMaint.

Troubleshooting JethroMaint issues

1. Verify that JethroMaint service is running: ***service jethro status***

2. If the service is not running, start it by running:
   service jethro restart
   service jethro status

   You can also run it only for specific single instance:
   service jethro start instance-name maint

3. If the service does not start, check its log at: *$JETHRO_HOME/instance/instance_name/log/services.log*
   Check the log for errors. The JethroMaint log is located at ***$JETHRO_HOME/instance/instance_name/log/jethro_maint.log***