Managing Apache Kudu

Date published:
Date modified:

https://docs.cloudera.com/

Legal Notice

Contents

Limitations

Review the server management and cluster management guidelines that you should consider before implementing Kudu.

Server management limitations

Here are some of the server management guidelines that you should consider before implementing Kudu.

- Production deployments should configure a least 4 GiB of memory for tablet servers, and ideally more than 16 GiB when approaching the data and tablet scale limits.
- Write ahead logs (WALs) can only be stored on one disk.
- Data directories cannot be removed. You must reformat the data directories to remove them.
- Tablet servers cannot be gracefully decommissioned.
- Tablet servers cannot change their address or port.
- Kudu has a hard requirement on having an up-to-date NTP. Kudu masters and tablet servers will crash when out of sync.
- Kudu releases have only been tested with NTP. Other time synchronization providers such as Chrony may not work.

Cluster management limitations

When managing Kudu clusters, review the following limitations and recommended maximum point-to-point latency and bandwidth values.

- Recommended maximum point-to-point latency within a Kudu cluster is 20 milliseconds.
- Recommended minimum point-to-point bandwidth within a Kudu cluster is 10 Gbps.
- If you intend to use the location awareness feature to place tablet servers in different locations, it is recommended that you measure the bandwidth and latency between servers to ensure they fit within the above guidelines.
- All masters must be started at the same time when the cluster is started for the very first time.

Start and stop Kudu processes

You can start, stop, and configure Kudu services to start automatilcally by using the CLI commands.

Start Kudu services using the following commands:

```
sudo service kudu-master start
sudo service kudu-tserver start
```

To stop Kudu services, use the following commands:

```
sudo service kudu-master stop
sudo service kudu-tserver stop
```

Configure the Kudu services to start automatically when the server starts, by adding them to the default runlevel.

```
sudo chkconfig kudu-master on            # RHEL / CentOS
sudo chkconfig kudu-tserver on           # RHEL / CentOS
sudo update-rc.d kudu-master defaults    # Ubuntu
sudo update-rc.d kudu-tserver defaults   # Ubuntu
```

Orchestrate a rolling restart with no downtime

Kudu 1.12 provides tooling to restart a cluster with no downtime. This topic provides the steps to perform rolling restart.

About this task

Note: If any tables in the cluster have a replication factor of 1, some quiescing tablet servers will never become fully quiesced, as single-replica tablets do not naturally relinquish leadership. If such tables exist, use the kudu cluster rebalance tool to move replicas of these tables away from the quiescing tablet server by specifying the --ignored_tservers, --move_replicas_from_ignored_tservers, and --tables options.

Note: If running with rack awareness, the following steps can be performed by restarting multiple tablet servers within a single rack at the same time. Use ksck to ensure that the location assignment policy is enforced while going through these steps, and that no more than a single location is restarted at the same time. At least three locations should be defined in the cluster to safely restart multiple tablet service within one location.

Cloudera Manager can automate this process, by using the “Rolling Restart” command on the Kudu service.

Note: Cloudera Manager does not support automatic moving of the single-replica tablets.

Cloudera Manager will prompt you to specify how many tablet servers to restart concurrently. If running with rack awareness with and at least three racks specified across all hosts that contain Kudu roles, it is safe to specify the restart batch with up to one rack at a time, provided the rack assignment policy is being enforced.

The following service configurations can be set to tune the parameters the rolling restart will run with:

- Rolling Restart Health Check Interval: the interval in seconds that Cloudera Manager will run ksck after restarting a batch of tablet servers, waiting for the cluster to become healthy.
- Maximum Allowed Runtime to Rolling Restart a Batch of Servers: the total amount of time in seconds Cloudera Manager will wait for the cluster to become healthy after restarting a batch of tablet servers, before exiting with an error.

Procedure

1. Restart the master(s) one-by-one. If there is only a single master, this may cause brief interference with on-going workloads.
2. Starting with a single tablet server, put the tablet server into maintenance mode by using the kudu tserver state e nter_maintenance tool.
3. Start quiescing the tablet server using the kudu tserver quiesce start tool. This signals Kudu to stop hosting leaders on the specified tablet server and to redirect new scan requests to other tablet servers.
4. Periodically run kudu tserver quiesce start with the --error_if_not_fully_quiesced option, until it returns success, indicating that all leaders have been moved away from the tablet server and that all on-going scans have completed.
5. Restart the tablet server.
6. Periodically run ksck until the cluster ireports a healthy status.
7. Exit maintenance mode on the tablet server by running kudu tserver state exit_maintenance. This allows new tablet replicas to be placed on the tablet server.
8. Repeat these steps for all tablet servers in the cluster.

Related Information

Changing directory configuration

Minimize cluster disruption during temporary planned downtime of a single tablet server

If a single tablet server is brought down temporarily in a healthy cluster, all tablets will remain available and clients will function as normal, after potential short delays due to leader elections. However, if the downtime lasts for more than --follower_unavailable_considered_failed_sec (default 300) seconds, the tablet replicas on the down tablet server will be replaced by new replicas on available tablet servers. This will cause stress on the cluster as tablets re-replicate and, if the downtime lasts long enough, significant reduction in the number of replicas on the down tablet server, which may require the rebalancer to fix.

To work around this, in Kudu versions 1.11 onward, the kudu CLI contains a tool to put tablet servers into maintenance mode. While in this state, the tablet server's replicas are not re-replicated due to its downtime alone, though re-replication may still occur in the event that the server in maintenance suffers from a disk failure or if a follower replica on the tablet server falls too far behind its leader replica. Upon exiting maintenance, re-replication is triggered for any remaining under-replicated tablets.

The kudu tserver state enter_maintenance and kudu tserver state exit_maintenance tools are added to orchestrate tablet server maintenance. The following can be run from a tablet server to put it into maintenance:

```
$ TS_UUID=$(sudo -u kudu kudu fs dump uuid --fs_wal_dir=<wal_dir> --fs_data_
dirs=<data_dirs>)
$ sudo -u kudu kudu tserver state enter_maintenance <master_addresses> "$T
S_UUID"
```

The tablet server maintenance mode is shown in the "Tablet Servers" page of the Kudu leader master's web UI, and in the output of kudu cluster ksck. To exit maintenance mode, run the following command:

```
sudo -u kudu kudu tserver state exit_maintenance <master_addresses> "$TS_UUI
D"
```

Kudu web interfaces

Kudu tablet servers and masters expose useful operational information on a built-in web interface.

Kudu master web interface

Kudu master processes serve their web interface on port 8051. The interface exposes several pages with information about the state of the cluster.

- A list of tablet servers, their host names, and the time of their last heartbeat.
- A list of tables, including schema and tablet location information for each.
- SQL code which you can paste into Impala Shell to add an existing table to Impala's list of known data sources.

Kudu tablet server web interface

Each tablet server serves a web interface on port 8050. The interface exposes information about each tablet hosted on the server, its current state, and debugging information about maintenance background operations.

Common web interface pages

Both Kudu masters and tablet servers expose the following information via their web interfaces:

- HTTP access to server logs.

- An /rpcz endpoint which lists currently running RPCs via JSON.
- Details about the memory usage of different components of the process.
- The current set of configuration flags.
- Currently running threads and their resource consumption.
- A JSON endpoint exposing metrics about the server.
- The version number of the daemon deployed on the cluster.

These interfaces are linked from the landing page of each daemon's web UI.

Best practices when adding new tablet servers

A common workflow when administering a Kudu cluster is adding additional tablet server instances, in an effort to increase storage capacity, decrease load or utilization on individual hosts, increase compute power, and more.

By default, any newly added tablet servers will not be utilized immediately after their addition to the cluster. Instead, newly added tablet servers will only be utilized when new tablets are created or when existing tablets need to be replicated, which can lead to imbalanced nodes. It's recommended to run the rebalancer CLI tool just after adding a new tablet server into the cluster.

Avoid placing multiple tablet servers on a single node. Doing so nullifies the point of increasing the overall storage capacity of a Kudu cluster and increases the likelihood of tablet unavailability when a single node fails (the latter drawback is not applicable if the cluster is properly configured to use the rack awareness (location awareness) feature.

To add additional tablet servers to an existing cluster, the following steps can be taken to ensure tablet replicas are uniformly distributed across the cluster:

1. Ensure that Kudu is installed on the new machines being added to the cluster, and that the new instances have been correctly configured to point to the pre-existing cluster. Then, start the new tablet server instances.
2. Verify that the new instances check in with the Kudu Master(s) successfully. A quick method for verifying whether they have successfully checked in with the existing Master instances is to view the Kudu Master WebUI, specifically the /tablet-servers section, and validate that the newly added instances are registered, and have a heartbeat.
3. Once the tablet server(s) are successfully online and healthy, follow the steps to run the rebalancing tool which spreads the existing tablet replicas to the newly added tablet servers.
4. After the rebalancer tool has completed, or even during its execution, you can check the health of the cluster using the ksck command-line utility.

Decommission or remove a tablet server

You can decommission or permanently remove a tablet server from a cluster.

About this task

Starting with Kudu 1.12, the Kudu rebalancer tool can be used to decommission a tablet server by supplying the --ig nored_tservers and --move_replicas_from_ignored_tservers arguments.

Note: Do not decommission multiple tablet servers at once. To remove multiple tablet servers from the cluster, follow the below instructions for each tablet server, ensuring that the previous tablet server is removed from the cluster and ksck is healthy before shutting down the next.

Procedure

1. Ensure the cluster is in good health using ksck.
2. Put the tablet server into a maintenance mode by using the kudu tserver state enter_maintenance tool.

3. Run the kudu cluster rebalance tool, supplying the --ignored_tservers argument with the UUIDs of the tablet servers to be decommissioned, and the --move_replicas_from_ignored_tservers flag.
4. Wait for the moves to complete and for ksck to show the cluster in a healthy state.

   Note: To verify that all the moves are completed from the server placed in maintenance mode, ensure that there are no replicas on the server by running kudu cluster ksck or running kudu cluster rebalancer with --report_only and --output_replica_distribution_details flags or checking the Web UI of the tablet server.
5. The decommissioned tablet server can be brought offline by stopping the tablet server in Cloudera Manager.
6. To completely remove it from the cluster so ksck shows the cluster as completely healthy, restart the masters.

   If you have only one master in your deployment, this may cause cluster downtime. In a multi-master deployment, restart the masters in sequence to avoid cluster downtime.

Related Information

Minimize cluster distruption during temporary planned downtime of a single tablet server

Use cluster names in the kudu command line tool

When using the kudu command line tool, it can be difficult to remember the precise list of Kudu master RPC addresses needed to communicate with a cluster, especially when managing multiple clusters. As an alternative, you can use the command line tool to identify clusters by name.

Procedure

1. Create a new directory to store the Kudu configuration file.
2. Export the path to this newly created directory in the KUDU_CONFIG environment variable.
3. Create a file called kudurc in the new directory.
4. Populate kudurc as follows, substituting your own cluster names and RPC addresses:

```
clusters_info:
  cluster_name1:
    master_addresses: ip1:port1,ip2:port2,ip3:port3
  cluster_name2:
    master_addresses: ip4:port4
```

5. When using the kudu command line tool, replace the list of Kudu master RPC addresses with the cluster name, prepended with the character @. For example:

```
$ sudo -u kudu kudu cluster @cluster_name1
```

   Note: Cluster names may be used as input in any invocation of the kudu command line tool that expects a list of Kudu master RPC addresses.

Migrate Kudu data from one directory to another on the same host

Take the following steps to move the entire Kudu data from one directory to another.

About this task

Note: The steps were verified on an environment where the master and the server instances were configured to write the WAL/Data to the same directory.

Procedure

1. Stop the Kudu service.
2. Modify the directory configurations for the Master/Server instances.
3. Move the existing data from the old directory, to the new one.
4. Make sure the file/directory ownership is set to the kudu user.
5. Restart the Kudu service.
6. Run ksck and verify for the healthy status.

Related Information

Changing directory configuration

Migrate to a multiple Kudu master configuration

Before migrating to a multiple Kudu masters set up, you need to perform many migration planning steps, such as deciding the number of masters, and choosing the nodes where to add the new Kudu masters.

About this task

The migration procedure does not require stopping all the Kudu processes in the entire cluster but once the migration is complete, all the Kudu processes must be restarted to incorporate the newly added masters. The restarting of the Kudu processes can be done without incurring downtime.

The procedure supports adding multiple masters at the same time by selecting multiple hosts.

Procedure

1. Decide how many masters to use.

   The number of masters should be odd because an even number of masters does not provide any benefit over having one fewer masters. Three or five node master configurations are recommended as they can tolerate the failure of one or two masters respectively.
2. Establish a maintenance window.

   One hour should be sufficient maintenance window time. During this time the Kudu cluster will be unavailable.
3. Configure a DNS alias for the new master or masters.

   The alias can be:

   - a CNAME record: if the machine already has an A record in DNS
   - and A record: if the machine is only known by its IP address

   

   Important: Without DNS aliases, it is not possible to recover from permanent master failures without restarting the masters and tablet servers in the cluster to pick up the replacement master node with a different hostname. It is highly recommended that you use DNS aliases.
4. Perform the following preparatory steps for each new master that you are planning to add:

   a) Choose a node in the cluster where there is no running Kudu master yet and which has enough spare CPU and memory capacity to host a new Kudu master.

   The master generates very little load so it can be collocated with other data services or load-generating processes, though not with another Kudu master from the same configuration.

   b) Choose and record the directories where the new master's data and WAL will be located.

   c) Choose and record the port the master should use for RPCs.

5. In Cloudera Manager, add a new Kudu Master role to the selected new master node:
   a) Select the Kudu service.
   b) Select Instances.
   c) Click Add Role Instances.
   d) Click Select hosts under Master × 1.
   e) Select one or multiple host node and click OK.
   f) Click Continue.
   g) Review the changes and if everything is correct, click Finish.
6. Start the newly added master role instance or instances.
7. Restart the Kudu service.
8. If you have Kudu tables that are accessed from Impala and you did not set up DNS aliases, update the HMS database manually in the underlying database that provides the storage for HMS:
   a) Connect to the HMS database.
   b) Run an SQL statement similar to the following example:

```
UPDATE TABLE_PARAMS
SET PARAM_VALUE =
  'master-1.example.com,master-2.example.com,master-3.example.com'
WHERE PARAM_KEY = 'kudu.master_addresses' AND PARAM_VALUE = 'master-1.e
xample.com';
```

   c) In impala-shell run the following command: INVALIDATE METADATA;

What to do next

To verify that all masters are working properly, consider performing the following checks:

- Using a browser, visit each master's web UI and navigate to the /masters page.

  All the masters should be listed there with one master in the LEADER role and the others in the FOLLOWER role. The contents of /masters on each master should be the same.
- Run a Kudu system check (ksck) on the cluster using the kudu command line tool.
- If applicable, run a few quick SQL queries against a couple of migrated Kudu tables using impala-shell or Hue.

Change master hostnames

When replacing dead masters, use DNS aliases to prevent long maintenance windows. If the cluster was set up without aliases, change the host names as described in this section.

Prepare for master hostname changes

In this step, you need to identify a down-time window, and note the UUID and the RPC address of each master.

Procedure

1. Establish a maintenance window during which the Kudu cluster will be unavailable. One hour should be sufficient.
2. On the **Masters** page in Kudu Web UI, note the UUID and RPC address of each master.
3. Stop all the Kudu processes in the cluster.
4. Set up the new hostnames to point to the masters and verify all servers and clients properly resolve them.

Perform master hostname changes

You need to bring the Kudu clusters down to update the master hostnames. Therefore, identify at least a one-hour maintenance window for this task.

Procedure

1. Rewrite each master's Raft configuration with the following command, executed on each master host:

   ```
   $ sudo -u kudu kudu local_replica cmeta rewrite_raft_config --fs_wal_dir
   =<master_wal_dir> [--fs_data_dirs=<master_data_dir>] 000000000000000000
   0000000000000 <all_masters>
   ```

   For example:

   ```
   $ sudo -u kudu kudu local_replica cmeta rewrite_raft_config --fs_wal_dir=/
   data/kudu/master/wal --fs_data_dirs=/data/kudu/master/data 000000000000000
   00000000000000000 4aab798a69e94fab8d77069edff28ce0:new-master-name-1:705
   1 f5624e05f40649b79a757629a69d061e:new-master-name-2:7051 988d8ac6530f42
   6cbe180be5ba52033d:new-master-name-3:7051
   ```

2. Update the master address:

   - In an environment not managed by Cloudera Manager, change the gflag file of the masters so the master_addre sses parameter reflects the new hostnames.
   - In an environment managed by Cloudera Manager, specify the new hostname in the Master Address (server.address) field on each Kudu role.

3. Change the gflag file of the tablet servers to update the tserver_master_addrs parameter with the new hostnames. In an environment managed by Cloudera Manager, this step is not needeed.
4. Start the masters.
5. To verify that all masters are working properly, perform the following checks:

   a) In each master's Web UI, click Masters on the Status Pages. All of the masters should be listed there with one master in the LEADER role field and the others in the FOLLOWER role field. The contents of Masters on all master should be the same.

   b) Run the below command to verify all masters are up and listening. The UUIDs are the same and belong to the same master as before the hostname change:

      ```
      $ sudo -u kudu kudu master list new-master-name-1:7051,new-master-name-2
      :7051,new-master-name-3:7051
      ```

6. Start all of the tablet servers.
7. Run a Kudu system check (ksck) on the cluster using the kudu command line tool. After startup, some tablets may be unavailable as it takes some time to initialize all of them.
8. If you have Kudu tables that are accessed from Impala, update the HMS database manually in the underlying database that provides the storage for HMS.

   a) The following is an example SQL statement you run in the HMS database:

      ```
      UPDATE TABLE_PARAMSSET PARAM_VALUE =
      'new-master-name-1:7051,new-master-name-2:7051,new-master-name-3:7051'
      WHERE PARAM_KEY = 'kudu.master_addresses'
      AND PARAM_VALUE = 'master-1:7051,master-2:7051,master-3:7051';
      ```

   b) In impala-shell, run:

      ```
      INVALIDATE METADATA;
      ```

   c) Verify updating the metadata worked by running a simple SELECT query on a Kudu-backed Impala table.

Related Information

Monitoring cluster health with ksck

Remove Kudu masters

In the event that a multi-master deployment has been overallocated nodes, the following steps should be taken to remove the unwanted masters.

Important:

- In planning the new multi-master configuration, keep in mind that the number of masters should be odd and that three or five node master configurations are recommended.
- Dropping the number of masters below the number of masters currently needed for a Raft majority can incur data loss. To mitigate this, ensure that the leader master is not removed during this process.

Prepare for removal

In order to remove the unwanted masters from a multi-master deployment, you need to identify them and note their UUID and RPC addresses.

Procedure

1. Establish a maintenance window (one hour should be sufficient). During this time the Kudu cluster will be unavailable.
2. Identify the UUID and RPC address current leader of the multi-master deployment by visiting the /masters page of any master's web UI. This master must not be removed during this process; its removal may result in severe data loss.
3. Stop the unwanted Kudu master processes.

Perform the removal

When you remove any Kudu masters from a multi-master deployment, you need to rewrite the Raft configuration on the remaining masters, remove data and WAL directories from the unwanted masters, and finaly modify the value of the tserver_master_addrs configuration parameter for the tablet servers to remove the unwanted masters. You need to bring the Kudu clusters down. Therefore, identify at least a one-hour maintenance window for this task.

Procedure

1. Perform the Raft configuration change by running the kudu master remove tool.

   Only a single master can be removed at a time. If multiple masters need to be removed, run the tool multiple times. In the following example master-2 is being removed from a Kudu cluster with two masters master-1, and master-2:

   ```
   $ sudo -u kudu kudu master remove master-1,master-2 master-2
   ```

2. Remove the data directories and WAL directory on the unwanted masters. This is a precaution to ensure that they cannot start up again and interfere with the new multi-master deployment.
3. Modify the value of the master_addresses configuration parameter for the masters of the new multi-master deployment.
4. Restart all of the masters that were not removed.
5. If you are not using Cloudera Manager: Modify the value of the tserver_master_addrs configuration parameter for the tablet servers to remove any unwanted masters.

6. Retart all of the tablet servers.

What to do next

To verify that all masters are working properly, consider performing the following checks:

- Using a browser, visit each master's web UI and navigate to the /masters page. All the masters should now be listed there with one master in the LEADER role and the others in the FOLLOWER role. The contents of /masters on each master should be the same.
- Run a Kudu system check (ksck) on the cluster using the kudu command line tool.

Related Information

Configure Kudu processes

Monitoring cluster health with ksck

Run the tablet rebalancing tool

The kudu CLI contains a rebalancing tool that can be used to rebalance tablet replicas among tablet servers. For each table, the tool attempts to balance the number of replicas per tablet server. It also, without unbalancing any table, attempts to even out the number of replicas per tablet server across the cluster as a whole.

The rebalancing tool should be run as the Kudu admin user, specifying all master addresses:

```
sudo -u kudu kudu cluster rebalance master-01.example.com,master-02.example.
com,master-03.example.com
```

When run, the rebalancer reports on the initial tablet replica distribution in the cluster, logs the replicas it moves, and prints a final summary of the distribution when it terminates:

```
Per-server replica distribution summary:
       Statistic       |   Value
-----------------------+-----------
 Minimum Replica Count | 0
 Maximum Replica Count | 24
 Average Replica Count | 14.400000
Per-table replica distribution summary:
 Replica Skew |  Value
--------------+----------
 Minimum      | 8
 Maximum      | 8
 Average      | 8.000000

I0613 14:18:49.905897 3002065792 rebalancer.cc:779] tablet e7ee9ade95b342a7a
94649b7862b345d: 206a51de1486402bbb214b5ce97a633c -> 3b4d9266ac8c45ff9a5d4d7
c3e1cb326 move scheduled
I0613 14:18:49.917578 3002065792 rebalancer.cc:779] tablet 5f03944529f44626
a0d6ec8b1edc566e: 6e64c4165b864cbab0e67ccd82091d60 -> ba8c22ab030346b4baa289
d6d11d0809 move scheduled
I0613 14:18:49.928683 3002065792 rebalancer.cc:779] tablet 9373fee3bfe74ce
c9054737371a3b15d: fab382adf72c480984c6cc868fdd5f0e -> 3b4d9266ac8c45ff9a5d4
d7c3e1cb326 move scheduled

... (full output elided)

I0613 14:19:01.162802 3002065792 rebalancer.cc:842] tablet f4c046f18b174cc
2974c65ac0bf52767: 206a51de1486402bbb214b5ce97a633c -> 3b4d9266ac8c45ff9a5d4
d7c3e1cb326 move completed: OK

rebalancing is complete: cluster is balanced (moved 28 replicas)
```

```
Per-server replica distribution summary:
       Statistic       |   Value
-----------------------+-----------
 Minimum Replica Count | 14
 Maximum Replica Count | 15
 Average Replica Count | 14.400000

Per-table replica distribution summary:
 Replica Skew |  Value
--------------+----------
 Minimum      | 1
 Maximum      | 1
 Average      | 1.000000
```

If more details are needed in addition to the replica distribution summary, use the --output_replica_distribution_details flag. If added, the flag makes the tool print per-table and per-tablet server replica distribution statistics as well.

Use the --report_only flag to get a report on table-wide and cluster-wide replica distribution statistics without starting any rebalancing activity.

The rebalancer can also be restricted to run on a subset of the tables by supplying the --tables flag. Note that, when running on a subset of tables, the tool does not attempt to balance the cluster as a whole.

The length of time rebalancing is run for can be controlled with the flag --max_run_time_sec. By default, the rebalancer runs until the cluster is balanced. To control the amount of resources devoted to rebalancing, modify the flag --max_moves_per_server. See kudu cluster rebalance --help for more.

It is safe to stop the rebalancer tool at any time. When restarted, the rebalancer continues rebalancing the cluster.

The rebalancer tool requires all registered tablet servers to be up and running to proceed with the rebalancing process in order to avoid possible conflicts and races with the automatic re-replication and to keep replica placement optimal for current configuration of the cluster. If a tablet server becomes unavailable during the rebalancing session, the rebalancer exits. As noted above, it is safe to restart the rebalancer after resolving the issue with unavailable tablet servers. However, if it is necessary to rebalance the cluster when a few tablet servers in a Kudu cluster are not available, it is possible to specify their UUIDs as a comma-separated list with the --\ignored_tservers flag and rebalance the rest of the cluster. With the --ignored_tservers flag, the specified tablet servers are effectively ignored by the rebalancer tool, which means they are not considered as a part of the cluster along with tablet replicas they host.

The rebalancing tool can rebalance Kudu clusters running older versions as well, with some restrictions. Consult the following table for more information. In the table, "RF" stands for "replication factor".

Version Range	Rebalances RF = 1 Tables?	Rebalances RF > 1 Tables?
v < 1.4.0	No	No
1.4.0 <= v < 1.7.1	No	Yes
v >= 1.7.1	Yes	Yes

If the rebalancer is running against a cluster where rebalancing replication factor one tables are not supported, it rebalances all the other tables and the cluster as if those singly-replicated tables did not exist.

Range-aware rebalancing

You can use the Kudu cluster rebalance CLI tool to run range-aware rebalancing. Range-aware rebalancing runs on a per-table basis, which means that the rebalancing can be run on one table at a time. To enable range-aware rebalancing for a particular table, you need to add the following flags when using the CLI tool:

- --enable_range_rebalancing
- --tables=<table_name_for_range_aware_rebalancing>

To perform cluster-wide rebalancing, it is recommended to run the kudu cluster rebalance tool. In addition to the cluster-wide rebalancing, you can run the range-aware rebalancing for larger tables in the cluster that are multilevel-partitioned, and can suffer from the hot-spotting issue.

Run a tablet rebalancing tool on a rack-aware cluster

It is possible to use the kudu cluster rebalance tool to establish the placement policy on a cluster. This might be necessary when the rack awareness feature is first configured or when re-replication violated the placement policy.

About this task

The rebalancing tool breaks its work into three phases:

Procedure

1. The rack-aware rebalancer tries to establish the placement policy. Use the ##disable_policy_fixer flag to skip this phase.
2. The rebalancer tries to balance load by location, moving tablet replicas between locations in an attempt to spread tablet replicas among locations evenly. The load of a location is measured as the total number of replicas in the location divided by the number of tablet servers in the location. Use the ##disable_cross_location_rebalancing flag to skip this phase.
3. The rebalancer tries to balance the tablet replica distribution within each location, as if the location were a cluster on its own. Use the ##disable_intra_location_rebalancing flag to skip this phase.

What to do next

By using the ##report_only flag, it's also possible to check if all tablets in the cluster conform to the placement policy without attempting any replica movement.

Run a tablet rebalancing tool in Cloudera Manager

You access and run the tablet rebalancing tool from Cloudera Manager.

Procedure

1. Browse to Clusters Kudu .
2. Click Actions and select Run Kudu Rebalancer Tool.

Results

In Cloudera Manager, the rebalancer runs with the default flags.

Managing Kudu tables with range-specific hash schemas

The custom hash partition feature enables you to vary the number of hash buckets per range partition, both at table creation and alteration time.

It is possible to specify a different number of hash buckets per range partition when creating or altering a table. This allows for accommodating new ranges of already existing Kudu tables to changes in workload patterns. For example, for a table range-partitioned by date or timestamp it makes sense to increase the number of hash buckets with an increase in ingestion rate to maximize write throughput.

When no hash schema per range is specified for a table's range either during creation or alteration time, it defaults to the table-wide hash schema. If no table-wide hash schema is specified when the table is created, no hash bucketing for the data is done at all.

Any hash schema specified for a range partition overrides the table-wide hash schema. However, the number of hash dimensions in any range-specific hash schema must be the same as in the table-wide hash schema.

The newly introduced range-specific hash schemas in Kudu are supported by impala-shell.

Range-specific hash schemas example: Using impala-shell

Review examples of using impala-shell to create and alter Kudu tables with range-specific hash schemas.

The following is a few examples of creating a new table with custom hash schemas for some of its range partitions.

The table-wide hash schema for table t1 has two hash dimensions (hash bucketing by the ‘id’ and the ‘name’ columns independently):

```
CREATE TABLE t1 (id INT, name STRING, PRIMARY KEY(id, name))
PARTITION BY HASH(id) PARTITIONS 3 HASH(name) PARTITIONS 4
RANGE (id)
(
    // hash partition by “id” and “name” as in the table-wide hash schema,
    // but customize the number of hash buckets in each dimension
    PARTITION 15 <= VALUES < 20 HASH(id) PARTITIONS 5 HASH(name) PARTITIONS
 3,
    // this range has the table-wide hash schema
    PARTITION 20 <= VALUES < 25
)
```

The table-wide hash schema for table t2 has one hash dimension (hash bucketing by the ‘name’ column):

```
CREATE TABLE t2 (id INT, name STRING, PRIMARY KEY(id, name))
PARTITION BY HASH(name) PARTITIONS 3
RANGE (id)
(
    // hash partition by “name” but use custom number of hash buckets
    PARTITION  5 <= VALUES < 10 HASH(name) PARTITIONS 5,

    // hash partition by “id”
    PARTITION 10 <= VALUES < 15 HASH(id) PARTITIONS 2,
    // hash partition by “id” and “name”
    PARTITION 15 <= VALUES < 20 HASH(id, name) PARTITIONS 8,

    // this range has the same schema as the range [15, 20) above:
    // hash partition by the primary key (i.e. by (id, name) pair) into 8
 buckets
    PARTITION 20 <= VALUES < 25 PARTITIONS 8,

    // using the table-wide hash schema if no override is specified
    PARTITION 25 <= VALUES < 30
)
```

The following are examples of adding new range partitions with custom hash schemas to a table that already exists:

```
// t2: add new range partition by “id”, hashed by “id” and “name” together
ALTER TABLE t2 ADD RANGE PARTITION -5 <= VALUES < 0 HASH PARTITIONS 10;

// t1: add new range partition by “id”, additionally hashed by “id” and “nam
e” separately
// (defaults to the table-wide hash schema since no range-specific hash sc
hema specified)
ALTER TABLE t1 ADD RANGE PARTITION 25 <= VALUES < 30
```

```
// t2: add new range partition by “id”, additionally hashed by “name”
ALTER TABLE t2 ADD RANGE PARTITION 30 <= VALUES < 40 HASH(name) PARTITIONS 5
;

// t1: add new range partition by “id”, additionally hashed by “id” and “
name” separately
ALTER TABLE t1 ADD RANGE PARTITION 40 <= VALUES < 50 HASH(id) PARTITIONS 10,
 HASH(name) PARTITIONS 5;
```

Range-specific hash schemas example: Using Kudu C++ client API

Review an example of using Kudu C++ client API to create a table with a range partition having range-specific hash schema and then alter the table, adding one more range with custom hash schema.

Note: After adding or dropping ranges with custom hash schemas you have to refresh the KuduTable handle.

```
 // Define the column schema for the table.
  KuduSchema schema;
  {
    KuduSchemaBuilder b;
    b.AddColumn("key")->Type(KuduColumnSchema::INT32)->NotNull()->PrimaryK
ey();
    b.AddColumn("int_col")->Type(KuduColumnSchema::INT32)->NotNull();
    b.AddColumn("string_col")->Type(KuduColumnSchema::STRING);
    RETURN_NOT_OK(b.Build(&schema));
  }
  unique_ptr<KuduTableCreator> tc(client->NewTableCreator());
  // The table is range-partitioned by the ‘key’ column and has table-wide
 hash schema:
  // one-dimensional hash bucketing into two buckets by the ‘key’ column.
  table_creator->table_name("test")
      .schema(&schema)
      .add_hash_partitions({ "key" }, 2)
      .set_range_partition_columns({ "key" });
  // Add [0, 100) range partition with custom hash schema:
  // 5 buckets with hash based on the "key" column with hash seed 8.
  {
    unique_ptr<KuduPartialRow> lower(schema.NewRow());
    RETURN_NOT_OK(lower->SetInt32("key", 0));

    unique_ptr<KuduPartialRow> upper(schema.NewRow());
    RETURN_NOT_OK(upper->SetInt32("key", 100));

    unique_ptr<KuduRangePartition> p(
        new KuduRangePartition(lower.release(), upper.release()));
    RETURN_NOT_OK(p->add_hash_partitions({ "key" }, 5, 8));
    tc->add_custom_range_partition(p.release());
  }

  // Create the ‘test’ table with the specified column schema
  // and one range partition.
  RETURN_NOT_OK(tc->Create());

  // Open the table and work with it.
  shared_ptr<KuduTable> table;
  RETURN_NOT_OK(client->OpenTable("test", &table));
  . . .
```

```
// Alter the table, adding a new range [100, 200) with range-specific hash
// schema: 3 buckets with hash based on the "key" column with hash seed 1.
{
  unique_ptr<KuduPartialRow> lower(schema.NewRow());
  RETURN_NOT_OK(lower->SetInt32("key", 222));

  unique_ptr<KuduPartialRow> upper(schema.NewRow());
  RETURN_NOT_OK(upper->SetInt32("key", 333));

  unique_ptr<KuduRangePartition> p(
      new KuduRangePartition(lower.release(), upper.release()));
  RETURN_NOT_OK(p->add_hash_partitions({ "key" }, 3, 1));

  unique_ptr<KuduTableAlterer> ta(client->NewTableAlterer(kTableName));
  ta->AddRangePartition(p.release());
  RETURN_NOT_OK(ta->Alter());
}

// NOTE: after adding or dropping a range with custom hash schemas
//       it's necessary to refresh the KuduTable handle
RETURN_NOT_OK(client->OpenTable("test", &table));

// Continue working with the table.
. . .
```

Range-specific hash schemas example: Using Kudu Java client API

Review an example of using Kudu Java client API to create a table with a range partition having range-specific hash schema and then alter the table, adding one more range with custom hash schema.

Note: After adding or dropping ranges with custom hash schemas you have to refresh the KuduTable handle.

```
// Define the column schema.
ArrayList<ColumnSchema> columns = new ArrayList<>(2);
columns.add(new ColumnSchema.ColumnSchemaBuilder(
    "key", Type.INT32).key(true).build());
columns.add(new ColumnSchema.ColumnSchemaBuilder(
    "col", Type.INT64).build());
final Schema schema = new Schema(columns);
CreateTableOptions builder = new CreateTableOptions();

// Do range partitioning on the "key" column.
builder.setRangePartitionColumns(ImmutableList.of("key"));
// Define the table-wide schema: two buckets on the "key" column.
builder.addHashPartitions(ImmutableList.of("key"), 2, 0);

// Add [0, 100) range partition with custom hash schema.
{
  PartialRow lower = schema.newPartialRow();
  lower.addInt("key", 0);
  PartialRow upper = schema.newPartialRow();
  upper.addInt("key", 100);

  // Custom hash schema for the range: five buckets on the "key" column.
  RangePartitionWithCustomHashSchema rangePartition =
      new RangePartitionWithCustomHashSchema(
          lower,
          upper,
```

```
            RangePartitionBound.INCLUSIVE_BOUND,
            RangePartitionBound.EXCLUSIVE_BOUND);
    rangePartition.addHashPartitions(ImmutableList.of("key"), 5, 0);
    builder.addRangePartition(rangePartition);
  }

  // Create the table.
  KuduTable table = client.createTable("mytable", schema, builder);

  // Work with the table.
  . . .

  // Alter the table: add [100, 200) range partition with custom hash sche
ma.
  {
    PartialRow lower = schema.newPartialRow();
    lower.addInt("key", 100);
    PartialRow upper = schema.newPartialRow();
    upper.addInt("key", 200);
    RangePartitionWithCustomHashSchema range =
        new RangePartitionWithCustomHashSchema(
            lower,
            upper,
            RangePartitionBound.INCLUSIVE_BOUND,
            RangePartitionBound.EXCLUSIVE_BOUND);
    range.addHashPartitions(ImmutableList.of("key"), 3, 0);
    client.alterTable("mytable",
        new AlterTableOptions().addRangePartition(range));
  }

  // NOTE: after adding or dropping a range with custom hash schemas
  //       it's necessary to refresh the KuduTable handle
  table = client.openTable("mytable");

  // Continue working with the table.
  . . .
```