Cloudera Runtime 7.1.7

Configuring Apache HBase High Availability

Date published: 2020-02-29
Date modified: 2021-07-30

CLOUDERA

https://docs.cloudera.com/

Legal Notice

Contents

Enable HBase high availability using Cloudera Manager....................................4

HBase read replicas...4

Timeline consistency..5

Keep replicas current..5

Read replica properties...5

Configure read replicas using Cloudera Manager...6

Using rack awareness for read replicas.. 7

Create a topology map.. 7

Create a topology script... 8

Activate read replicas on a table.. 8

Request a timeline-consistent read.. 8

Enable HBase high availability using Cloudera Manager

You can configure one or more backup Master role in Cloudera Manager to ensure that every component is highly available.

About this task

HBase administrators can configure HBase clusters with High Availability, or HA. You must configure HA when HBase applications require low-latency queries and can tolerate minimal (near-zero-second) staleness for read operations. Examples include queries on remote sensor data, distributed messaging, object stores, and user profile management.

Most aspects of HBase are highly available in a standard configuration. A cluster typically consists of one Master and three or more RegionServers. To ensure that every component is highly available, configure one or more backup Masters. The backup Masters run on other hosts than the active Master.

Procedure

1. In Cloudera Manager, select the HBase service.
2. Follow the process for adding a role instance and add a backup Master to a different host than the one on which the active Master is running.

HBase read replicas

You can configure read replicas to enable the HMaster to distribute read-only copies of regions (replicas) to different RegionServers in the cluster.

Without read replicas, only one RegionServer services a read request from a client, regardless of whether RegionServers are colocated with other DataNodes that have local access to the same block. This ensures consistency of the data being read. However, a RegionServer can become a bottleneck due to an underperforming RegionServer, network problems, or other reasons that could cause slow reads. With read replicas enabled, the HMaster distributes read-only copies of regions (replicas) to different RegionServers in the cluster. One RegionServer services the default or primary replica, which is the only replica which can service write requests. If the RegionServer servicing the primary replica is down, writes will fail. Other RegionServers serve the secondary replicas, follow the primary RegionServer and only see committed updates. The secondary replicas are read-only, and are unable to service write requests.

The secondary replicas can be kept up to date by reading the primary replica's HFiles at a set interval or by replication. If they use the first approach, the secondary replicas may not reflect the most recent updates to the data when updates are made and the RegionServer has not yet flushed the memstore to HDFS. If the client receives the read response from a secondary replica, this is indicated by marking the read as "stale". Clients can detect whether or not the read result is stale and react accordingly. Replicas are placed on different RegionServers, and on different racks when possible. This provides a measure of high availability (HA), as far as reads are concerned.

If a RegionServer becomes unavailable, the regions it was serving can still be accessed by clients even before the region is taken over by a different RegionServer, using one of the secondary replicas. The reads may be stale until the entire WAL is processed by the new RegionServer for a given region. For any given read request, a client can request a faster result even if it comes from a secondary replica, or if consistency is more important than speed, it can ensure that its request is serviced by the primary RegionServer. This allows you to decide the relative importance of consistency and availability, in terms of the CAP Theorem, in the context of your application, or individual aspects of your application, using Timeline Consistency semantics.

Timeline consistency

You can change your consistency level to TIMELINE to allow a more flexible standard of consistency than the default HBase model of STRONG consistency.

A client can indicate the level of consistency it requires for a given read (Get or Scan) operation. The default consistency level is STRONG, meaning that the read request is only sent to the RegionServer servicing the region. This is the same behavior as when read replicas are not used. The other possibility, TIMELINE, sends the request to all RegionServers with replicas, including the primary. The client accepts the first response, which includes whether it came from the primary or a secondary RegionServer. If it came from a secondary, the client can choose to verify the read later or not to treat it as definitive.

Keep replicas current

You can choose from two different mechanisms for keeping replicas up to date: using a timer or using replication.

Using a Timer: In this mode, replicas are refreshed at a time interval controlled by the configuration option hbase.regionserver.storefile.refresh.period.

Using Replication: In this mode, replicas are kept current between a source and sink cluster using HBase Async WAL replication. This works similarly to HBase’s multi-datacenter replication, but instead the data from a region is replicated to the secondary regions. Each secondary replica always receives and observes the writes in the same order that the primary region committed them. In some sense, this design can be thought of as *in-cluster replication*, where instead of replicating to a different datacenter, the data goes to secondary regions to keep secondary region’s in-memory state up-to-date. The data files are shared between the primary region and other replicas, so that no extra storage overhead exists. However, the secondary regions contain recent non-flushed data in their memstores, which increases the memory overhead. The primary region writes flush, compaction, and bulk load events to its WAL as well, which are also replicated through WAL replication to secondaries. When they observe the flush or compaction or bulk load event, the secondary regions replay the event to pick up the new files and drop the old ones.

Committing writes in the same order as in primary ensures that the secondaries do not diverge from the primary regions data, however; the data might still be stale in secondary regions because the log replication is asynchronous.

Async WAL Replication is disabled by default. You can enable this feature by setting hbase.region.replica.replication.enabled to true.

Read replica properties

You must understand the various properties to configure to enable support for read replicas in HBase.

Table 1: HBase Read Replica Properties

Property Name	Default Value	Description
hbase.region.replica.replication.enabled	false	The mechanism for refreshing the secondary replicas. If set to false, secondary replicas are not guaranteed to be consistent at the row level. Secondary replicas are refreshed at intervals controlled by a timer (hbase.regionserver.storefile .refresh.period), and so are guaranteed to be at most that interval of milliseconds behind the primary RegionServer. Secondary replicas read from the HFile in HDFS, and have no access to writes that have not been flushed to the HFile by the primary RegionServer. If true, replicas are kept up-to-date using replication.
hbase.regionserver.storefile.refresh.period	0 (disabled)	The period, in milliseconds, for refreshing the store files for the secondary replicas. The default value of 0 indicates that the feature is disabled. Secondary replicas update their store files from the primary RegionServer at this interval. If refreshes occur too often, this can create a burden for the NameNode. If refreshes occur too infrequently, secondary replicas will be less consistent with the primary RegionServer.
hbase.ipc.client.specificThreadForWriting	true	Whether or not to enable interruption of RPC threads at the client. This is required for region replicas with fallback RPC’s to secondary regions. The default value of true enables primary RegionServers to access data from other regions' secondary replicas.
hbase.client.primaryCallTimeout.get	10000 µs	The timeout period, in microseconds, an HBase client waits for a response before the read is submitted to a secondary replica if the read request allows timeline consistency. The default value is 10000 µs. Lower values increase the number of remote procedure calls while lowering latency.
hbase.client.primaryCallTimeout.multiget	10000 µs	The timeout period, in milliseconds, before an HBase client's multi-get request, such as HTab le.get(List<GET>)), is submitted to a secondary replica if the multi-get request allows timeline consistency. Lower values increase the number of remote procedure calls while lowering latency.

Configure read replicas using Cloudera Manager

You can configure read replicas using Cloudera Manager.

Procedure

1. Before you can use replication to keep replicas current, you must set the column attribute REGION_MEMST ORE_REPLICATION to false for the HBase table, using HBase Shell or the client API.
2. In Cloudera Manager, select the HBase service.
3. Click the Configuration tab.
4. Select Scope > HBase or HBase Service-Wide .
5. Select Category > Advanced .
6. Locate the HBase Service Advanced Configuration Snippet (Safety Valve) for hbase-site.xml property or search for it by typing its name in the Search box.

7. Create a configuration and paste it into the text field. The following example configuration demonstrates the syntax:

```
<property>
  <name>hbase.regionserver.storefile.refresh.period</name>
  <value>0</value>
</property>
<property>
  <name>hbase.ipc.client.allowsInterrupt</name>
  <value>true</value>
  <description>Whether to enable interruption of RPC threads at the client
. The default value of true is
    required to enable Primary RegionServers to access other RegionServers
 in secondary mode. </description>
</property>
<property>
  <name>hbase.client.primaryCallTimeout.get</name>
  <value>10</value>
</property>
<property>
  <name>hbase.client.primaryCallTimeout.multiget</name>
  <value>10</value>
</property>
```

8. Click Save Changes to commit the changes.
9. Restart the HBase service.

Using rack awareness for read replicas

Rack awareness for read replicas is modeled after the mechanism used for rack awareness in Hadoop.

The default implementation, which you can override by setting hbase.util.ip.to.rack.determiner, to custom implementation, is ScriptBasedMapping, which uses a topology map and a topology script to enforce distribution of the replicas across racks.

Create a topology map

The topology map assigns hosts to racks. You can create a topology map using Cloudera Manager.

About this task

The topology map assigns hosts to racks. It is read by the topology script. A rack is a logical grouping, and does not necessarily correspond to physical hardware or location. Racks can be nested. If a host is not in the topology map, it is assumed to be a member of the default rack. The following map uses a nested structure, with two data centers which each have two racks. All services on a host that are rack-aware will be affected by the rack settings for the host.

Procedure

- If you use Cloudera Manager, do not create the map manually. Instead, go to Hosts, select the hosts to assign to a rack, and select Actions for Selected > Assign Rack .

```
<topology>
  <node name="host1.example.com" rack="/dc1/r1"/>
  <node name="host2.example.com" rack="/dc1/r1"/>
  <node name="host3.example.com" rack="/dc1/r2"/>
  <node name="host4.example.com" rack="/dc1/r2"/>
  <node name="host5.example.com" rack="/dc2/r1"/>
  <node name="host6.example.com" rack="/dc2/r1"/>
  <node name="host7.example.com" rack="/dc2/r2"/>
  <node name="host8.example.com" rack="/dc2/r2"/>
</topology>
```

Create a topology script

The topology script determines rack topology using the topology map.

About this task

The topology script determines rack topology using the topology map. By default, CDH uses /etc/hadoop/conf.cloudera.YARN-1/topology.py

Procedure

- To use a different script, set net.topology.script.file.name to the absolute path of the topology script.

Activate read replicas on a table

After enabling read replica support on your RegionServers, configure the tables for which you want read replicas to be created.

About this task

After enabling read replica support on your RegionServers, configure the tables for which you want read replicas to be created.

Procedure

- At table creation, to create a new table with read replication capabilities enabled, set the REGION_REPLICATION property on the table. Use a command like the following, in HBase Shell:

  ```
  hbase> create 'myTable', 'myCF', {REGION_REPLICATION => '3'}
  ```

- By altering an existing table, you can also alter an existing column family to enable or change the number of read replicas it propagates, using a command similar to the following. The change will take effect at the next major compaction.

  ```
  hbase> disable 'myTable'
  hbase> alter 'myTable', 'myCF', {REGION_REPLICATION => '3'}
  hbase> enable 'myTable'
  ```

Request a timeline-consistent read

You can use request a timeline-consistent read in your application.

About this task

To request a timeline-consistent read in your application, use the get.setConsistency(Consistency.TIMELINE) method before performing the Get or Scan operation.

To check whether the result is stale (comes from a secondary replica), use the isStale() method of the result object. Use the following examples for reference.

Procedure

- Get request

  ```
  Get get = new Get(key);
  get.setConsistency(Consistency.TIMELINE);
  Result result = table.get(get);
  ```

- Scan request

  ```
  Scan scan = new Scan();
  scan.setConsistency(CONSISTENCY.TIMELINE);
  ResultScanner scanner = table.getScanner(scan);
  Result result = scanner.next();
  ```

- Scan request to a specific replica.

  This example overrides the normal behavior of sending the read request to all known replicas, and only sends it to the replica specified by ID.

  ```
  Scan scan = new Scan();
  scan.setConsistency(CONSISTENCY.TIMELINE);
  scan.setReplicaId(2);
  ResultScanner scanner = table.getScanner(scan);
  Result result = scanner.next();
  ```

- Detect a stale result

  You can also request timeline consistency using HBase Shell, allowing the result to come from a secondary replica.

  ```
  hbase> get 'myTable', 'myRow', {CONSISTENCY => "TIMELINE"}
  hbase> scan 'myTable', {CONSISTENCY => 'TIMELINE'}
  ```