HBase Backups Architecture

Backups are needed as insurance against a few data loss/corruption scenarios:

1. Data loss due to some code bug
2. Data loss due to hardware failure
3. Data loss due to operator error
4. Data corruption due to HBase/HDFS bug
5. Data corruption due to bugs in application using HBase

Backup guarantees:

* #1, #2 and #4
  + very rare events
  + cluster should be properly provisioned
  + properly monitored
  + code being used is properly tested
  + restore to the latest data possible to minimize data loss
* #3
  + operator errors
  + eg. – accidentally deleting data in a production DFS cluster by a script that was not written correctly
  + restore to the latest data possible to minimize data loss
* #5
  + application had a bug which corrupted data
  + eg. – application bug caused inconsistent data writes
  + restore to data before the corruption started

In order to cover most failure scenarios, we need 3 stages of backups, defined below:

* Stage 1 – in-Rack Backup Unit
  + Backup stored on the live DFS used by HBase
  + Protects against error types #3 (where the HBase directory is deleted on the live DFS) and #5
* Stage 2 – in-DataCenter Backup Unit
  + Backup on another DFS in the same DC
  + Protects against #3 (some or all data deleted on live DFS), #5
  + Protects in a limited way against #1, #2, #4
* Stage 3 – cross-DataCenter Backup Unit
  + Backed up on another DFS in a different DC
  + Protects against DC level failures

Features supported by the various backup stages:

|  |  |  |  |
| --- | --- | --- | --- |
| **Protection against** | **Stage 1** | **Stage 2** | **Stage 3** |
| Operator errors | X | X | X |
| Fast Restores | X | X |  |
| Stable Storage |  | X | X |
| Heterogeneous Racks |  | X | X |
| Datacenter Separation |  |  | X |

Some requirements for backups:

* Preserve row-key atomicity
* Cannot only backup HLogs – need reasonably fast restores
* All the data requested should be backed up successfully
* Ability to backup multiple cf’s

How incremental backups work at the Stage 1 level:

1. Periodically take snapshots of the HFiles
   1. List all regions
   2. For each region, ask the RS hosting it for a list of HFiles
   3. Copy the HFiles to the backup location
   4. If HFiles were deleted (due to compactions) try to reclaim from .Trash
   5. If HFiles were not in .Trash fail the region – it will be retried
   6. Make sure all regions succeed for snapshot to succeed
2. The listing of files in #1 is not atomic across column-families, so we could get inconsistent set of files. In order to preserve per-row atomicity, we need to replay HLog edits for the duration of #1
3. Copy HLogs frequently
   1. The frequency of copying HLogs will determine the data-loss window.
   2. If we want to guarantee we wont lose more than 1 hour of data, we need to copy HLogs every hour.
4. To restore to a point in time, choose appropriate HFile backup and play HLogs forward to desired point in time
   1. The backups rely on the clocks across the various region-servers for determining the point in time to which the edits are re-played.
   2. The hlog files are of the format hlog.TIMESTAMP, TIMESTAMP is time when log is created. We look at this time to determine the file set. We need all files where TIMESTAMP > start time and TIMESTAMP < finish time. We need the latest file where TIMESTAMP < start time.
   3. The idea here is to prepare the finalized table in a temp location by copying/moving the data (HFiles + HLogs) into the right directory structures, and add it into the Hbase cluster. Then let distributed log splitting replay the edits files upon enabling the table.

Notes/FAQ about stage 1 backup:

Q: Why can we not lock flushes while we copy the HFiles to get atomicity?

The current design allows this backup to be performed against a remote HDFS cluster, as well as for large data sizes when the copy of files could take a long time.

Q: Why not just take a copy of the directory of a table or all tables?

Two reasons:

* In some cases, we only want to backup a subset of column families (and omit some derived data).
* A second reason is that we issue a flush (this is an option) before listing the files so that we can be sure we have all the data as of the backup time. Since there is no way to determine when a flush finishes, we do this inside the RS.

Q: D. Import the directory into the running HBase with META entries, etc (this already exists)". Is the tool that exists "completebulkload"?

No this is a custom tool, which moves/copies the backed up table from the backup location into the Hbase table and fixes-up META accordingly. The region names also have to be altered in case the exported table's name is different from the table name into which we are restoring.

For Stage 2 and stage 3, we have 2 dials we can tune but the basic backup strategy is essentially the same:

1. Keep copying HLogs frequently
2. Keep copying HFile backups (less frequently)

For example, in the CBU case #1 can per every hour and #2 twice week. For DBU, #1 could be every hour and #2 every month.

Tools

1. Backup one, multiple or all CF’s in a table
2. Backing up to Stage 1, Stage 2 and Stage 3
3. Retain only last n versions of the backups
4. Restore backups to a point in time
   1. Restore to a running or offline cluster
   2. Restore to the same table name or a new one
5. Restore tables on the Stage 2 as needed
6. Verify a percentage of data in the backups (this may not be do-able without app knowledge)
7. Alerts and dashboards for monitoring