HBase Backup and Restore

IBM: (Demaj Ni, Richard Ding, Jerry He), sorry if I forgot somebody :)
Hortonworks Inc.: Vladimir Rodionov (vrodionov@hortonworks.com)

Table of contents:

Revision history:

Author	Date	Comment	Version
Richard Ding	March 31, 2014	initial version	0.1
Richard Ding	June 4, 2014	V2	0.2
Vladimir Rodionov	June 29, 2015	V3	0.3
Vladimir Rodionov	July 1st, 2015	first feedback from HW team	0.4

Background and requirements

The proposed design is based on a work done by Demaj Ni, Jerry He, Richard Ding and others (IBM) and submitted in a series of patches to Apache HBase repository. The master JIRA ticket is HBASE-7912 (HBase Backup/Restore based on snapshots), two other worth mentioning are: HBASE-10900 (Full backup) and HBASE-11085 (Incremental backup restore support). There are other tickets (subtasks), most of them are in an open state and have no patches:

HBASE-11172 Cancel a backup process
HBASE-11173 Show Backup History
HBASE-11174 show backup/restore progress
HBASE-11175 improve Backup/Restore framework by abstracting out zookeeper
HBASE-11180 Merge backups
HBASE-11182 Store backup information in a manifest file using protobuffer format
HBASE-11181 prune/delete old backups
HBASE-10900 FULL table backup and restore

There have been attempts in the past to come up with a viable HBase backup/restore solution (e.g., HBASE-4618). Recently, there are many advancements and new features in HBase, for example, FileLink, Snapshot, and Distributed Barrier Procedure. This is a proposal for a backup/restore solution that utilizes these new features to achieve better performance and consistency.

HBase Backup and restore is a crucial requirement by enterprises using HBase as data repository. Some solutions have been attempted in the past, for example, solutions using

HBase Export and Import API. Given the availability of some important new HBase features, notably HBase snapshot (HBASE-6055, HBASE-7290), we propose a new HBase Backup and Restore design based on these new features. The benefit is ease of use and better performance and accuracy. Backup and restore is used for data recovery in case of data loss or failures. HBase snapshot enables the efficient and effective capture of a consistent and point-in-time (both to some degree) HBase table image. But snapshot alone is not enough for a complete backup and restore/recovery solution. The snapshot information and data is tightly coupled and stored with the existing HBase cluster -- in-place backup, besides this, HBase does not support incremental snapshots, which makes only snapshots less than optimal solution for a data backup. We want to backup HBase data to FileSystem across clusters and possibly to other storage media or servers. This proposal is a logical extension to the snapshot feature.

Key features and Use Cases

A common practice of backup and restore in database is to first take full baseline backup, and then periodically take incremental backup that capture the changes since the full baseline backup. HBase cluster can store massive amount data. Therefore we want use full backup in combination with incremental backups for HBase as well.

The following is a typical use case scenario for full and incremental backup:

- The user takes a full backup of a table or a set of tables in HBase.
- The user schedules periodical incremental backups to capture the changes from the full backup, or from last incremental backup.
- The user needs to restore table data to a past point in time.
- The full backup is restored to the table(s) or to different table name(s). Then the incremental backups that are up to the desired point in time are applied on top of the full backup.

We would support the following key features and capabilities.

- Backup to DFS FileSystem across clusters and possibly to other storage media or servers.
- Support single table or a set of tables backup and restore (full and incremental).
- Restore to different table names and to different clusters.

- Supports adding and removing tables to and from backup set without interruption of incremental backup schedule.
- Supports merging of incremental backups into longer period and bigger incremental backups for easy storage and restore.
- Supports scheduled backups.
- Unified command line interface for all the above.

To illustrate these key capabilities, the following are two more detailed use case examples.

Use case example 1:

1. User takes a full backup of a set of tables (i.e. table1 and table2) in HBase.
2. User takes incremental backups. The incremental backup will only track table1 and table2.
3. User adds other tables (i.e. table3 and table4) in HBase, and an implicit full backup is executed during the add process
4. User continues to take incremental backups. The incremental backup data would cover table1, table2, table3 and table4.
5. User wants to restore table3 and table4 to a past PIT (point-in-time).
6. Full backup in 3. is restored onto HBase cluster. Then the incremental backups after that full backup are applied on top of the full restore until the PIT.

Use case example 2:

1. User takes a full backup of a set of tables in HBase.
2. User takes daily incremental backups.
3. User merges the daily incremental backups into weekly incremental backups.
4. User combines/rolls up the weekly incremental backup into monthly incremental backups.
5. User wants to restore the tables to a past PIT.
6. Full backup is restored onto HBase cluster.
7. Monthly incremental backups before the desired PIT are applied.
8. Closest daily backups up to the PIT are applied.

Overall design

Design summary

The solution covers the two main parts of backup and restore.

- **Full backup and restore**. Backup will first invoke HBase snapshot and export snapshot internally. The full backup can be restored with HBase bulk import utility.
- **Incremental backup** uses WALs to capture the data changes since last full backup or incremental backup. We execute roll log across region servers to track the WALs that need to be in the backup. Then a distributed copy is used to move the physical files to target FileSystem. The WALs can be replayed into HFiles on the fly or offline. During restore, the HFiles will be applied on top of the full backup via HBase Bulk Load utility.

Client Side

The client initiates request to start a full backup or incremental backup via a command line interface. The typical input of this request is:

1. Type (full or incremental)
2. The table names (in a format namespace:name, namespace:* will add all tables in this namespace) or backup set name (see backup set management below).
3. The directory location where the backed-up files are to be copied.
4. Other options.

Client synchronously waits for the completion (either successful or failed) of this request. Asynchronous execution mode will also be provided. In case of async execution, command exits immediately after request submission, later on user can check status and progress of a backup operation using CLI tool.

Backup Manager

The Backup client passes the backup request to the Backup Manager. Backup manager checks the type of the backup request and delegates accordingly, and act as a singleton to make sure only one backup is ongoing (*vrodionov: this requirement will be relaxed in next release*). Backup Manager can also do other bookkeeping work or cleanup work as needed. The Backup Manager will pass the request to Backup Handler for execution of the backup.

Tracking Progress and Aborting

The user should be able to track the progress of the backup process. The user should be able to abort the process if the user chooses to do so, for example, in the case of a performance drag. The user should be able to track the restore progress as well and abort restore progress when necessary.

Backup Manifest, other meta information and history

We want to have a manifest file for each backup. The manifest file will have such information as the type of the backup, the size, the location info, etc. The manifest will also provide the dependency lineage info needed to restore the backup. For example, an incremental restore will need to know the required full backup or incremental backups that are needed before it can be applied.
We want to keep all table meta-information (table and region infos) inside backup image directory.
All information, which is needed to restore table(s), MUST be stored in HDFS to make possible restore operation to another cluster when primary cluster is unavailable.

Backup sets management

User can create, delete, modify backup sets, using CLI tool. Backup set uniquely identifies set of HBase tables with a single name. User can add or remove a table (or tables) to or from backup set at a point in time. The following commands will be supported:

1. backup_add ‘setname’ ‘table1’ table2’ ...
2. backup_remove ‘setname’ ‘table3’
3. backup_createset ‘setname’
4. backup_deleteset ‘setname’
5. backup_listsets

When a table (or tables) is added to backup set, we will do an initial full backup for the table(s), and any incremental backup after that point in time will include the table(s). When a table (or tables) is removed from backup set, any incremental backup after that point in time will filter out the table(s) if needed. Adding or removing a table to or from backup set will not alter incremental backup sequence for existing tables tracked for incremental backup that are not part of the add/remove. That means later incremental backup will not be from this add/remove action to the current for these tables. Later incremental backup will still from last incremental backup to current.

hbase:backup table

This system table will keep track of all backup sessions:

- Write/Read backup session state.
- Write/Read backup session progress (per region server).
- Write/Read restore session progress.
- Stores last backuped WAL file timestamp (per region server).
- Stores list of all backed up WAL files (for **BackupLogCleaner**).
- Stores backup sets

This system table MUST be backed up and restored separately from other tables. This table stores information, which relevant only for backup sessions and CLI tools (backup management utils). All information, which is needed to **restore** tables is stored in HDFS to make restore possible to another cluster when primary cluster is down.

Full Backup

Sequence

1. BackupClient issues full backup request either through HBaseAdmin API or by using CLI tool.
2. BackupManager dispatches request to BackupHandler.
3. BackupHandler kicks off distributed log roll procedure through HBaseAdmin API.
4. HBase Master executes procedure across the cluster.
5. RS receives and executes roll log procedure and records last log file timestamp into **hbase:backup** table.
6. BackupHandler waits for procedure completion, then updates backup progress in **hbase:backup** table.
7. We need to verify the log procedure result. (check # of active RS before and after)
8. BackupHandler, for every table from backup set, executes taking snapshot (by means of HBaseAdmin API) followed by copying snapshot (using modified version of ExportSnapshot M/R job, see BackupCopier class). Network IO throttling can be enforced during copy operation.
9. BackupHandler finalizes backup (deletes snapshot, updates **hbase:backup**, stores manifest file and table region infos in backup destination directory).

Take snapshot

We generate a snapshot request by using a snapshot name suffixed with the current timestamp. For example, 'backup_ 1360652124199'. We then invoke the Snapshot Manager to take the snapshot and wait for its completion status.

Backup from existing Snapshot

We can allow the use of an existing snapshot by allowing the user to give a snapshot name. In this case, no new snapshot will be taken. The existing snapshot will be verified to exist and then the data will be copied to the target backup location. This feature will help to convert existing snapshots into valid backup images.

From full backup to incremental backup

We use WALs to track incremental changes. An important step is a clean cut on what need to be included in the next incremental backup after a full backup. We will do a log roll on each RS, and each RS will record its current WAL number. The information will be saved for later incremental backup. ~~This current WAL will be included in the next incremental backup files, but only the content after the recorded WAL number will be used.~~ In the next incremental backup, only WAL files with timestamp larger than recorded in a previous full or incremental backup will be included in a backup set.

Export Snapshot

We will invoke HBaseAdmin export snapshot call to copy out the full backup data. Network IO can be throttled during copy operation to guarantee that regular HBase operations are not affected seriously by backup.

Full restore

The backed-up HFiles will be restored using HBase bulk load utility. Tables can be created if not exist. ~~**hbase:backup** is restored in a temp table, then rows from this temp table override rows in existing **hbase:backup (we add missing rows from temp to hbase:backup, existing rows from temp will override those in hbase:backup)**. TBD (to be discussed how to backup/restore backup meta info)~~

hbase:backup table is always backuped and restored in a separate sessions, because it is a small table, there is no need for incremental backup support (incremental flag will be ignored).

1. hbase backup hbase:system
2. hbase restore hbase:system

Incremental Backup

Sequence

1. BackupClient issues incremental backup request either through HBaseAdmin API or by using CLI tool.
2. BackupManager dispatches request to BackupHandler.
3. BackupHandler kicks off distributed log roll procedure through HBaseAdmin API.
4. HBase Master executes procedure across the cluster.
5. RS receives and executes roll log procedure and records last log file timestamp into **hbase:backup** table.
6. BackupHandler waits for procedure completion, then updates backup progress in **hbase:backup** table.
7. BackupHandler identifies list of WAL files to be copied using info from **hbase:backup** table, then kicks off DistCp M/R job to copy files over to backup destination. Network IO throttling can be enforced during copy operation.
8. BackupHandler finalizes backup (deletes snapshot, updates **hbase:backup** and snapshots it, snapshotid of **hbase:backup** and time is attached to backup manifest, stores manifest file in backup destination directory).

Some design details of original (IBM) implementation

When client issues an incremental backup request, BackupManager will check the request and then kicks off a global procedure via HBaseAdmin for all the active regionServer to roll log. Each Region server will record its log number into ~~zookeeper~~ **hbase:backup**. Then we determine which log needs to be included in this incremental backup, and use DistCp to copy them to target location. At the same time, a dependency of backup image will be recorded, and later on saved in Backup Manifest file.

Restore is to replay the backuped WAL logs on target HBase instance. The replay will occur after full backup.

An incremental backup image depends on prior full backup image and incremental images if exists. Manifest file will be used to store the dependency lineage during backup, and used during restore time for PIT restore.

Converting Incremental WALs into HFiles and Incremental Restore

We'll convert/replay the backed-up WALs into HFiles for fast incremental restore. This will done offline without impacting the running HBase instance. When we replay the WALs, we will only filter to include the tables that we need to include ***(vrodionov: this is not implemented)***. Therefore only the HFiles that cover the backed-up tables are stored as incremental backup.

Merging Backups

After converting incremental WALs to HFiles, we will have a tool to merge incremental backups or/and full backups. The HFiles from incremental backup will be combined to form a bigger longer period incremental backup image. e.g. daily incremental backup can be combined to form weekly incremental backup. Incremental backups can also be merged with their parent full backup. The incremental backup images after convert and merge will be restored to a cluster via HBase bulk load.

Security

Security MUST be supported. One of the options - only authorized user (GLOBAL ADMIN) must be allowed to perform backup/restore/merge/delete and there should be ACLs on backup info read operations: history, status, progress, list, describe etc. See: HBASE-7367 for good discussion on snapshot security model. The single admin approach may not work well in a multi tenant environment (see below).

Multi Tenancy support

TBD (To Be Discussed). Tenants/namespaces will be supported in a future releases. We would like to avoid delegating admin rights on a per table basis.

Detail layout and frame work (from HBASE-10900)

The patch is a wrapper of the existing snapshot and exportSnapshot, and will use as the base framework for the overall solution of HBase-7912 as described below:

Note: TBI - to be implemented.

- **bin/hbase** : end-user command line interface to invoke BackupClient and RestoreClient
- **BackupClient.java** : 'main' entry for backup operations. This patch only supports 'full' and 'incremental' backup and restore. In future jiras, will support:
    - **create** incremental backup
    - **cancel** an ongoing backup/restore
    - **delete** an existing backup image
    - **describe** the detailed information of backup image
    - show **history** of all successful backups
    - show the **status** of the latest backup request
    - **convert** incremental backup WAL files into HFiles. either on-the-fly during create or after create
    - **merge** backup image
    - **stop/resume** backup a table of existing backup image
    - **show** tables of a backup image

- **BackupCommands.java** : a place to keep all the command usages and options
- **BackupManager.java** : handle backup requests on client-side, ~~create BACKUP ZOOKEEPER nodes to keep track backup. The timestamps kept in zookeeper will be used for future incremental backup (not included in this jira)~~, creates record in **hbase:backup** table to keep track backup session, creates BackupContext instance and dispatches backup request to BackupHandler.
- **BackupHandler.java** : in this patch, it is a wrapper of snapshot and exportsnapshot.
    - **timestamps** info will be recorded ~~in ZK~~ in **hbase:backup** table (TBI).
    - carries on **incremental** backup.
    - updates backup **progress in hbase:backup (TBI)**
    - sets flags of **status**
    - builds up **backupManifest** file(in this jira only limited info for fullback. later on, timestamps and dependency of multiple backup images are also recorded here)
    - cleans up after **failed** backup
    - cleans up after **cancelled** backup (TBI)
    - allows on-the-fly **convert** during incremental backup (TBI)
- **BackupContext.java** : encapsulate backup information like backup ID, table names, directory info, phase, TimeStamps of backup progress, size of data, ancestor info, etc.
- **BackupCopier.java** : the copying operation. Later on, to support progress report and mapper estimation; and extends DistCp for progress updating to **hbase:backup** during backup.
- **BackupException.java**: to handle exception from backup/restore
- **BackupManifest.java** : encapsulate all the backup image information. The manifest info will be bundled as manifest file together with data. So that each backup image will contain all the info needed for restore.
- **BackupStatus.java** : encapsulate backup status at table level during backup progress
- **BackupUtil.java** : utility methods during backup process
- **RestoreClient.java** : 'main' entry for restore operations. There is only one restore operation, which accepts ‘backupid’ as a parameter. So we can restore tables to a particular point in time (not consistent across the cluster, of course) by providing particular backupId (each is timestamped).
- **RestoreUtil.java** : utility methods during restore process
- **SnapshotCopy.java** : only a wrapper at this moment. But will be extended to keep track progress(maybe should implemented in ExportSnapshot directly?)
- **BackupRestoreConstants.java** : add the constants used by backup/restore code.
- **HBackupFilesystem.java** : the filesystem related api used by BackupClient and RestoreClient.
- **BackupLogCleaner.java** : log cleaner plugin, which overrides default log cleaner logic.

Feature development roadmap

Phase 0 - Design finalization (PH0)

1. Update original design document (IBM)
2. Discuss internally @HW HBase group
3. Publish updated doc @Apache HBase
4. Collect feedback and finalize the design.

Phase 1 - Reengineering (PH1) (preliminary list of work items) V 0.5

Functional version of Backup/Restore with limited functionality.

Note: Work items, which can be started before PH0 is complete marked as (!)

Note: 'Abstract' means - provide interfaces/abstract classes and framework to instantiate concrete implementations.

1. **Abstract DistCp** (incremental backup) to support non-M/R based implementations. Provide M/R implementation. DistCp is used to copy WAL files during incremental backup. (!)
2. **Abstract SnapshotCopy** (full backup) to support non-M/R based implementations. Provide M/R implementation. SnapshotCopy is used to copy snapshot's data during full backup operation (!)
3. **Abstract WALPlayer** (incremental restore) to support non-M/R based implementations. Provide M/R implementation
4. **Abstract Coordination manager** (Zk) operations. See org.apache.hadoop.hbase.coordination package for references. Provide M/R implementation. (!)
5. **hbase:backup** - move all backup meta info from Zk (coordination manager) to hbase system table. Do not use Zk (coordination manager) as a persistent storage.
6. **Custom WAL archive cleaner (BackupLogCleaner)**. We need to keep WAL files in archive until they either gets copied over to backup destination during an incremental backup or full backup (for ALL tables) happens. This is tricky, but is doable. Backup-aware WAL archive cleaner should consult **hbase:backup** to determine if WAL file is safe to purge.
7. **BUG**: deletion of a table with backup history (has Zk node) results in RuntimeException on all subsequent backup requests. See: BackupClient.requestBackup.
8. **BUG**: during incremental backup, provided table list is ignored and replaced with the set of tables, which have been already backuped before. Test case: backup T1, T2, T3, then request incremental backup for T1, T2 => T3 will be included as well. See: BackupClient.requestBackup. (!)
9. **TASK**: BackupHandler.deleteSnapshot MUST use HBase API for that (HBaseAdmin) - not direct FS access (deleting snapshot folder may be not enough?). (!)
10. **TASK**: move distributed log roll procedure call to BackupHandler.call from IncrementalBackupManager.getLogFilesForNewBackup. (!)
11. **TASK**: Make list of tables in **restore** command optional. If missing - all tables from backup image MUST be restored. Currently, one has to specify: backup root dir, backupId and list of tables (not very convenient). (!)

Phase 2 - Feature set enhancements (PH2) V1.0

Fully functional version.

1. **Backup management**. The following operations need to be supported:
    a. **cancel** an ongoing backup/restore
    b. **delete** an existing backup image
    c. **describe** the detailed information of backup image
    d. show **history** of all successful backups
    e. show the **status** of the latest backup request
    f. **convert** incremental backup WAL files into HFiles. Either on-the-fly during create or after create
    g. **merge** backup image. What is the difference between merge and convert? Merge works only on converted files (in original design), but we can relax this requirement. We can merge backup images as long as they are of the same type (HFiles or WALs).
    ~~h. **stop/resume** backup a table of existing backup image~~
    i. **show** tables of a backup image
    j. **progress** show progress of a backup by a given backup id.
    k. **schedule** - backup scheduling.
    l. **backup sets** management.

Note: stop/resume backup functionality will not be supported in a first releases.

2. **Backup throttling**. ExportSnapshot/DistCp supports IO throttling per map task - this needs to be exposed to backup utility command line tool. Backups must not interfere with regular HBase cluster operations.
3. **Restore throttling**. Do we need restore throttling as a feature?
4. **Security**. Security is not supported. Only authorized user (GLOBAL ADMIN) must be allowed to perform backup/restore. See: HBASE-7367 for good discussion on snapshot security model. Multi-tenancy? Table/namespace admin. Should we delegate privilege to NS admin? Do we have NS admin?
5. **Backup from existing snapshot**. I do see use case here, when backup is enabled first time and there are already point-in-time snapshots for table(s), which need to be converted into backups.
6. **Enhance HBaseAdmin API** to include backup/restore - related API.
7. **TASK: Avoid WAL files duplication** during incremental backup. Currently, if we perform incremental backup separately for, say, two tables, WAL files will be duplicated in a backup site due to the fact that the path to WAL includes backupId and every backup will have its separate copies of WAL files. Having centralized storage for WAL files on a backup site will improve performance and storage usage, especially for use cases where backup operations are finely grained (not global).

8. **TASK**: **(mutually exclusive with 9.) Filter WALs on backup to include only edits from backup set tables. Either 9 or 10 should be implemented.**

Phase 3 - Performance optimization (PH3) - V2.0

Performance optimized version.

1. **Improve WALPlayer**. Current patch restores one table at a time. This can be improved by changing WALPlayer code to work with multiple tables. In this case we won't to run M/R job for every table - only one M/R. This is M/R implementation - specific feature.
2. **Improve WALPlayer more**. Incremental restore runs WALPlayer in direct mode (loads edits directly to HBase table), this can be changed to create HFiles instead and use bulk load utility to load files into table regions directly.
3. **Enhance ExportSnapshot (M/R job)** to support multiple tables/snapshots. Currently, we do full backup exporting one table at a time. There is a separate M/R Job for every table in a backup set.

Phase 4 More features (PH4) - V3.0

1. **Multi-tenancy support**
2. **Backup API REST/Thrift access**. Read only? Full privilege? May interfere with HBase security.
3. **Multiple backup/restore sessions support**. TBD (not in a first version).
4. **Non M/R implementation of DistCp**
5. **Non M/R implementation of SnapshotCopier (ExportSnapshot)**
6. **Non M/R implementation of WALPlayer.**