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
Vladimir Rodionov	July 2, 2015		0.5
Vladimir Rodionov	July 6, 2015		0.6
Vladimir Rodionov	March 29, 2016	Added Appendix ABC	0.7

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. This proposal is a logical extension to the snapshot feature (and replacement of in many cases).

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.
- Support adding and removing tables to and from backup set without interruption of incremental backup schedule.
- Support merge of incremental backups into longer period and bigger incremental backups for easy storage and restore.
- Support 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

See Appendix A. B. C. on backup destination file system layout, description of backup manifest and backup image structure.

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. Client 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 a 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 session will be included in a backup set.

Incremental snapshot or incremental backup?

If we have already full table snapshots why would not we add incremental feature to a snapshot itself?

1. Snapshots are lightweight by design, they just record and store table metadata and list of store files, there is no need for incremental support unless you do export operation.
2. You will need incremental snapshots only if you export snapshot to another location, but this is what exactly backup is for - cloning and copying table data to another location.
3. This will duplicate available features to end users: they will be able to clone and copy table using either snapshot tool or backup tool.
4. The only user of Snapshot incremental API is going to be a backup tool.
5. We prefer to use backup/restore tools and terminology because it is widely adopted - not snapshots and snapshot tools.
6. Moving this code into snapshots will introduce large change to the existing HBase code base, while backup/restore feature barely touches existing HBase code.

7. Snapshots are good for quick *in-place* backups (w/o actual data copying) and should be used only for that purpose. They are especially good in QA environment, where quick table rollback feature is important. Lightweight snapshot-style backups can be introduced as well in a future.

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

Snapshot optimization

The current snapshot implementation is suboptimal from the performance point of view. First of all, for every region in a table it performs memstore flush, then it creates a large number of small HFileLink files on HDFS. Both these steps take considerable amount of time. Even with flush disabled, for large clusters, taking snapshot take minutes to complete. From mailing list:

> We do SKIP_FLUSH. We have 1200+ regionservers with a single table with 60k regions and 4 column families. It takes around 30 minutes to snapshot this table.

The reason: a gigantic amount of small HFileLink files which need to be created ~(60K x 4 x 10) = 2.4M.

One approach to snapshot is to record the following (per Region Server):

1. List of store files per region
2. List of wal files in WALs directory

3. max sequenceID from store files
4. current sequenceID at the time of snapshot.
5. reduce # of new files on HDFS : one manifest file per region/cf or even one manifest file per table
6. update special HBase table to keep hfile and wal links in there
7. have special archiver plugins for both: hfiles and wals.

During backup copy, we first copy hfiles over to new destination, for every table in backup set and copy WAL files with filtering minSeq <= seqId <= currentSeq at the end.

Advantage:

1. We do not do memstore flushes
2. We do not do log rolls
3. We do not create large # of files on HDFS
4. Somehow we do not have to flood HBase as well. (???)

Incremental Backup

Sequence

1. Client 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 their log number into ~~zookeeper~~ **hbase:backup**. Then we determine which log need 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.

Incremental Restore

Overall design

Restore is to replay the backuped WAL logs on target HBase instance. An incremental backup image depends on prior full backup image and incremental images if exists. For every table from incremental backup image, first full backup image will be identified and restored followed by all intermediate incremental backup images between the image being restored and last full backup image. Manifest file will be used to store the dependency lineage during backup, and used during restore time for PIT restore.

Sequence

1. Client issues incremental restore request either through HBaseAdmin API or by using CLI tool and provides backup image id.
2. For every table from a given backup image RestoreClient :
    a. Loads manifest file
    b. Identifies dependence list - list of backup images, this backup image depends from. First image in this list MUST be full backup image.
    c. It starts restore from first full backup image
    d. Then perform restore of incremental backup images one-by-one.
    e. For every incremental backup RestoreClient runs WALPlayer tool and loads WALs into a table.

3. RestoreClient finalizes restore (must update status of operation in **hbase:backup** table).

First incremental after full backup restore

As since we do not record last backed up sequence ids (MVCC) and do not restore up to that sequence id - that is kind of tricky, there will be some duplicates of KVs in store files after first incremental restore after full backup. These duplicates are result of how we do full backup and first incremental backup after full one. During full backup we perform distributed log roll and record, for every RS, last WAL timestamp, then we do snapshot. The next WAL after recorded one will make it into a next incremental backup set, but it will contains some edits (puts, deletes) which have been recorded by a previous snapshot. During restore, we, first, restore snapshot, then we will re-play WALs and this operation can create some duplicates of KVs in different store files.

Cell (KVs) deduplication can be added to StoreScanner. This is a minor change. **This is deduplication during read mode.**

Another possible approach is to record maximum sequence ID after full backup. Sequence ID is per Region and WAL is per RegionServer. We will need to store maximum sequence id per region after full backup. When we finish snapshot, we can collect all maximum sequence ids from store files and store them as a Map<Region, long>. During first incremental restore, we will need to check sequence id of every WAL Entry. If it is below or equals to a maximum seq id for this region - skip this entry. The logic will remains unchanged for all other incremental restores after the first one. **This is deduplication during restore mode.**

Deduplication during backup requires filtering WAL files during copy operation and may be implemented if we will support WAL files filtering.
So, the deduplication can be implemented either during read (original approach mentioned in the doc) or during restore (see above) or even during backup. Deduplication during backup will require WAL filtering during copy support, but this is a feature which is on the roadmap. There are some pro- and contra- for all of them.

READ: very simple to implement, but we will have duplication of some data in a file system of HBase after restore.
RESTORE: not that simple, but no data duplication in HBase cluster after restore, but there will be some data duplication in backup location.
BACKUP: not that simple as well, but no duplication at all ... but relies on WAL filtering during copy support.

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. The will include 10-12 JIRA tickets, most of them 1-2 days of work.

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. (!) - 1 day
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 (!) - 1 day
3. **Abstract WALPlayer** (incremental restore) to support non-M/R based implementations. Provide M/R implementation (!) - 1 day
4. **Abstract Coordination manager** (Zk) operations. See org.apache.hadoop.hbase.coordination package for references. Provide M/R implementation. (!) - 1 day
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. - 2 days
6. **Custom WAL archive cleaner (BackupLogCleaner)**. We need to keep WAL files in archive until they either get 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 archiver cleaner should consult **hbase:backup** to determine if WAL file is safe to purge. (!) 2 days

7. **BUG**: deletion of a table with backup history (has Zk node) results in RuntimeException on all subsequent backup requests. See: BackupClient.requestBackup. (!) - 0.5 day
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. (!) - 0.5 day
9. **TASK**: BackupHandler.deleteSnapshot MUST use HBase API for that (HBaseAdmin) - not direct FS access (deleting snapshot folder may be not enough?). (!) - 0.5 day
10. **TASK**: move distributed log roll procedure call to BackupHandler.call from IncrementalBackupManager.getLogFilesForNewBackup. (!) 0.5 day
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).~~ (!) 0.5 day

Total estimate: 10.5 days (2 weeks)

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.

There will be approximately 12 JIRA tickets (one per command or command group) Convert/Merge will take 1 week at least - 5-7 days, backup sets and schedule - 3-5 days, all others 0.5-1 days of work each.

Total estimate: up to 3 weeks (15 days) but 'convert' may require more time.

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. 1 day
3. **Restore throttling**. Do we need restore throttling as a feature? XXX
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 days
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. 0.5 day
6. **Enhance HBaseAdmin API** to include backup/restore - related API. 2-3 days
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). *2 days*
8. **TASK**: **(mutually exclusive with 7.) Filter WALs on backup to include only edits from backup set tables. Either 7 or 8 should be implemented. 2 days**
9. **BUG**: KeyValue deduplication must be added to StoreScanner. See "First incremental after full backup restore" sub-section.
10. **OR** (see 9): we need to take care of deduplication during restore (check "First incremental backup section after full backup")
11. **OR**: (see 9, 10) we need to take care of deduplication during backup. This depends on a feature from

Total estimate: 8 weeks.

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.**
7. **Multiple RS per host (?)**

Critical bugs

Below are the list of BUGs which can affect Snapshot/Backup/Restore functionality, stability, performance:

1. "Zk watches leak during snapshots": https://issues.apache.org/jira/browse/HBASE-13885

Appendix A. Backup root directory structure and layout

Full Backup Example

Backup Root Directory (ROOT): hdfs://localhost:57762/backupUT
Table namespace: default
Table name: test-1459375580152
BackupId : backup_1459375618126

ROOT/backup_1459375618126/default/test-1459375580152/.backup.manifest
ROOT/backup_1459375618126/default/test-1459375580152/.hbase-snapshot/snapshot_14593
75618910_default_test-1459375580152/.snapshotinfo
ROOT/backup_1459375618126/default/test-1459375580152/.hbase-snapshot/snapshot_14593
75618910_default_test-1459375580152/data.manifest
ROOT/backup_1459375618126/default/test-1459375580152/archive/data/default/test-1459375
580152/543f8c02c388dd931fb9bcd1c38e7372/f/a6ce4789f9b444d89bbc755254afd27d

Incremental Backup Example

BackupId : backup_1459378688723

Incremental backup of 3 tables: test-1459378649438, test-14593786494381,
test-14593786494382

Basically, we store only backup manifest (see Appendix B), table descriptor and region infos.

ROOT/backup_1459378688723/default/test-1459378649438/.backup.manifest
ROOT/backup_1459378688723/default/test-1459378649438/.tabledesc/.tableinfo.0000000001
ROOT/backup_1459378688723/default/test-1459378649438/fb2760e626d1b24a7e55fa37dfab4
608/.regioninfo
ROOT/backup_1459378688723/default/test-14593786494381/.backup.manifest
ROOT/backup_1459378688723/default/test-14593786494381/.tabledesc/.tableinfo.0000000001
ROOT/backup_1459378688723/default/test-14593786494381/90044069d5409273df8695e257d
f3c0a/.regioninfo
ROOT/backup_1459378688723/default/test-14593786494382/.backup.manifest
ROOT/backup_1459378688723/default/test-14593786494382/.tabledesc/.tableinfo.0000000001
ROOT/backup_1459378688723/default/test-14593786494382/3c722c6af580cfa06b6db005907
4b743/.regioninfo

WAL files are kept here:

ROOT/backup_1459378688723/WALs/.backup.manifest
ROOT/backup_1459378688723/WALs/10.22.11.177,58809,1459378626079/10.22.11.177%2C58809%2C1459378626079.1459378659124
ROOT/backup_1459378688723/WALs/10.22.11.177,58812,1459378626279/10.22.11.177%2C58812%2C1459378626279.1459378659124

Appendix B. Backup Manifest

Protobuf structure:

```
message BackupManifest {
  required string version = 1;                      -   Always 1.0
  required string backup_id = 2;                    - backup ID
  required BackupType type = 3;                     -FULL,INCREMENTAL
  repeated TableName table_list = 4;          -list of tables in
backup
  required uint64 start_ts = 5;               - self descriptive
  required uint64 complete_ts = 6;            - self descriptive
  repeated TableServerTimestamp tst_map = 7;  - see below
  repeated BackupImage dependent_backup_image = 8; - see below
}
```

tst_map - Incremental backup timestamp map

`Map<TableName, Map<String, Long>>` - for every table from table_list (4) it keeps map of RegionServers and last recorded WAL timestamp. This information is used during next incremental backup session to compile list of new WAL files.

dependent_backup_image - backup image of this backup session (See Appendix C.)

Appendix C. Backup image

The only purpose of BackupImage data structure is to keep lineage of all previously created backup images, which we need to restore all the tables from a current backup set.

Each backup image has a list of tables and backup id, as well as list of `ancestors`.

Protobuf structure:

```
message BackupImage {
  required string backup_id = 1;
  required BackupType backup_type = 2;
  required string root_dir = 3;
  repeated TableName table_list = 4;
  required uint64 start_ts = 5;
  required uint64 complete_ts = 6;
  repeated BackupImage ancestors = 7;
}
```

Algorithm: To build list of dependent images

```
Input: List<TableName> table_list
Output: List<BackupImage> ancestors: empty
```

Get full history of backup sessions from hbase:backup table in descending order (recent comes first):

List<BackupCompleteData> history = getBackupHistory();

for(BackupCompleteData backup: history) {
If (backup does not intersects table_list) continue;
if(backup is FULL and not covered by `ancestors`) {
Get list of tables from this backup and remove its intersection from `table_list`
Add BackupImage from this backup to `ancestors`
} else { // INCREMENTAL
Add BackupImage only if: it intersects original table_list and not covered completely by existing images in `ancestors`
}
}

Covered by `ancestors`:

The image list can cover the image only when the following conditions are satisfied:

1. each image from the list must not be an incremental image;
2. each image from the list must be taken after image has been taken;
3. union of table sets of the list must cover the table set of the image.

Appendix D. Backup command line tool

Usage

Usage: **hbase backup COMMAND**
  where COMMAND is one of:
    create create a new backup image;
    cancel cancel an ongoing backup;
    delete delete an existing backup image;
    describe show the detailed information of a backup image;
    history show history of all successful backups;
    progress show the progress of the latest backup request;
    set backup set management;

  Enter: **hbase backup help COMMAND** - to see help message for each command;

To run **hbase** go to $HBASE_HOME/bin directory.

Create backup command

  Usage: **hbase backup create** <type> <backup_root_path> [tables] [-s name]
      [-silent] [-w workers][-b bandwidth]
      type "full" to create a full backup image,
        "incremental" to create an incremental backup image;
      backup_root_path The full root path to store the backup image,
        the prefix can be hdfs, webhdfs, gpfs, s3fs;
      tables If no tables are specified, all tables are backed up,
        otherwise it is a comma separated list of tables. Namespace wildcards are
          not supported yet. If you want to backup the whole namespace you will
have to provide full list of tables.
      -s name Use the specified snapshot for full backup\n"
      -w number of parallel workers to copy data to backup destination.
      -b bandwidth per one worker (in MB sec)

Progress command

Usage: **hbase backup progress** <backupId>
    backupId backup image id;

Shows progress of a backup session.

Describe command

Usage: **hbase backup describe** <backupId>
backupId backup image id;

Describes backup image.

History command

Usage: **hbase backup history [-n N]**
-n N show up to N last backup sessions, default - 10;

Show last N backup sessions. If number of sessions is not specified, the command shows - 10.

Delete command

Usage: **hbase backup delete** <backupId>
backupId backup image id;

Deletes backup image from the system.

Backup Set command

Usage: **hbase set SUBCOMMAND [name] [tables]**
name Backup set name;
tables If no tables are specified, all tables will belong to the set,
Otherwise it is a comma separated list of tables;
where SUBCOMMAND is one of:
add add tables to a set, crete set if needed
remove remove tables from set
list list all sets
describe describes set
delete delete backup set

Backup sets management. Users can group tables into named backup sets and use set's names instead of table list names in backup commands, which accepts list of tables.

Restore backup

Usage: **hbase restore <backup_root_path> <backup_id> <tables> [tableMapping] [-overwrite] [-check] [-automatic]**

backup_root_path The parent location where the backup images are stored

backup_id The id identifying the backup image

table(s) Table(s) from the backup image to be restored.
Tables are separated by comma. TODO - support for sets

Options:

tableMapping A comma separated list of target tables.
If specified, each table in <tables> must have a mapping.

-overwrite With this option, restore overwrites to the existing table "
if there's any in restore target. The existing table must be online before restore.

-check With this option, restore sequence and dependencies are checked and verified without executing the restore\n"

-automatic With this option, all the dependencies are automatically restored together with this backup image following the correct order.
The restore dependencies can be checked by using -check option,
or using \"hbase backup describe\" command. Without this option,
only this backup image is restored;

Restore backup image.