Timeline Service V2's native HBase Tables

Introduction

This document is a proposal for the native HBase tables to accommodate the aggregation. The "real-time" aggregation should be based strictly on the native HBase schema. Any monitoring use cases that rely on the real-time aggregation data should also be based strictly on the native HBase schema.

Design considerations

(in order of importance):

1. Must be able to deal with scale and volume during writes
2. Must avoid full table scans on reads. Each scan must have a row-key limit, preferably in addition to a columnValueFilter.
3. Multiple writes are preferred over reads-on-write. This allows buffered writes and avoids needing to flush writes before reads.
4. Writes should ideally be idempotent, so in case of partial errors, we can retry.
5. We should be able to load historical data out of order. Ideally we should not rely on the time when writes appear to the back-end for correctness (without a way to override the time that would normally default)

Tables

entity table

- Stores entity details (exactly same as existing timeline entity table in the current hbase writer)
- Per App collector writes to it

- Connects to several region servers
- primary key: user ! cluster ! flow ! run id ! application id ! entity type ! entity id
- Every entity write is stored here, except for the entityType= yarn_application, those will be stored in the application table.
- Example table:
-
-

-

```
* entity_table
*|-------------------------------------------------------------|
*|  Row       | Column Family    | Column Family | Column Family |
*|  key       | info             | metrics       | config        |
*|-------------------------------------------------------------|
*| userName!  | id:entityId      | metricName1:  | configKey1:   |
*| clusterId! |                  | metricValue1  | configValue1  |
*| flowId!    | type:entityType  | @timestamp1   |               |
*| flowRunId! |                  |               | configKey2:   |
*| AppId!     | created_time:    | metricName1:  | configValue2  |
*| entityType!| 1392993084018    | metricValue2  |               |
*| entityId   |                  | @timestamp2   |               |
*|            | modified_time:   |               |               |
*|            | 1392995081012    | metricName2:  |               |
*|            |                  | metricValue1  |               |
*|            | r!relatesToKey:  | @timestamp2   |               |
*|            | id3!id4!id5      |               |               |
*|            |                  |               |               |
*|            | s!isRelatedToKey:|               |               |
*|            | id7!id9!id5      |               |               |
*|            |                  |               |               |
*|            | e!eventKey:      |               |               |
*|            | eventValue       |               |               |
*|            |                  |               |               |
*|            | flowVersion:     |               |               |
*|            | versionValue     |               |               |
*|-------------------------------------------------------------|
```

application table

- Stores application level details (summary of application metrics, application lifecycle events, application config)
- Nearly identical to the entities table but split for the applications for performance reasons (basically split from the entities table)
- RM's collector and Per App collector write to it
- From RM, connection is made to one or very few region servers.
- primary key: cluster ! user ! flow ! run id ! application id

flow_run table

- Stores per flow run information aggregated across applications, flow version
- RM's collector writes to on app creation and app completion
- Per App collector writes to it for metric updates at a slower frequency than the metric updates to application table
- primary key: cluster ! user ! flow ! flow run id
- Only the latest version of flow-level aggregated metrics will be kept, even if the entity and application level keep a timeseries.
- The running_apps column will be incremented on app creation, and decremented on app completion.
- For min_start_time the RM writer will simply write a value with the tag for the applicationId. A coprocessor will return the min value of all written values. Upon flush and compactions, the min value between all the cells of this column will be written to the cell without any tag (empty tag) and all the other cells will be discarded.
- Ditto for the max_end_time, but then the max will be kept.
- Tags are represented as #type:value. The type can be not set (0), or can indicate running (1) or complete (2). In those cases (for metrics) only complete app metrics are collapsed on compaction.
- The m! values are aggregated (summed) upon read. Only when applications are completed (indicated by tag type 2) can the values be collapsed.
- The application ids that have completed and been aggregated into the flow numbers are retained in a separate column for historical tracking: we don't want to re-aggregate for those upon replay

```
* flow_run table
* |-----------------------------------------|
* |  Row key   | Column Family              |
* |            | info                       |
* |-----------------------------------------|
* | clusterId! | flow_version:version7       |
* | userName!  |                            |
* | flowId!    | running_apps:1             |
* | flowRunId  |                            |
* |            | min_start_time:1392995080000 |
* |            | #0:""                      |
* |            |                            |
* |            | min_start_time:1392995081012 |
* |            | #0:appId2                  |
* |            |                            |
* |            | min_start_time:1392993083210 |
* |            | #0:appId3                  |
* |            |                            |
* |            |                            |
* |            | max_end_time:1392993084018 |
* |            | #0:""                      |
* |            |                            |
* |            |                            |
* |            | m!mapInputRecords:127      |
* |            | #0:""                      |
* |            |                            |
* |            | m!mapInputRecords:31       |
* |            | #2:appId2                  |
* |            |                            |
* |            | m!mapInputRecords:37       |
* |            | #1:appId3                  |
* |            |                            |
* |            |                            |
* |            | m!mapOutputRecords:181     |
* |            | #0:""                      |
* |            |                            |
* |            | m!mapOutputRecords:37      |
* |            | #1:appId3                  |
* |            |                            |
* |            |                            |
* |-----------------------------------------|
```

app_flow table

- Stores application to flow mapping
- RM's collector writes to it on the app creation
- Used to efficiently look up flow and flow-run given only an application ID (for example, for the read path).
- primary key: cluster ! application id

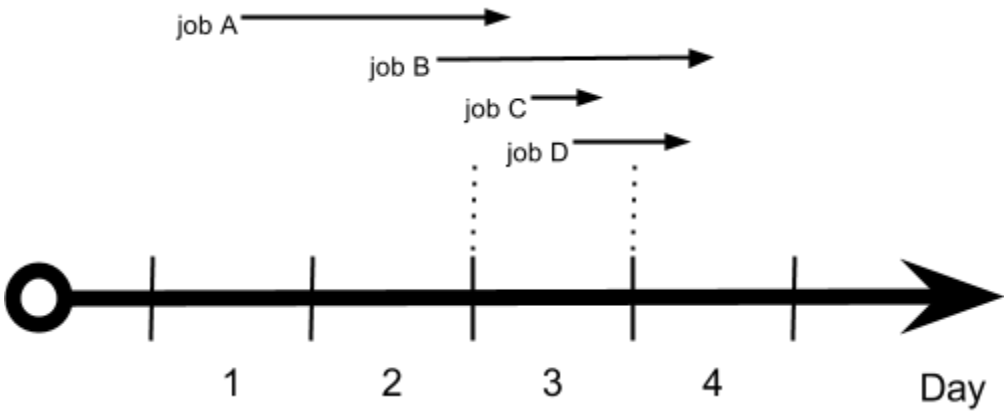
flow_version table

- Stores the unique versions per flow
- Helps to determine when was the first time a version was seen
- Helps to determine which new flows showed up on the cluster for the first time in given time range
- primary key: cluster! user ! flow ! version
- Column r!runId. Upon app start, the RM collector can simply write runId with a tag type 1 (indicating min version) and appID for the tag value. Then we can have a co-processor that will return the min version. Flush and compaction can simply chuck any value not the min (whether apps are complete or not).
- Ditto algorithm for s!flow_run_start

flow_activity table

- Stores per day flow run pointers and info
- Written to by RM's collector for application lifecycle
- primary key: cluster ! day timestamp ! user ! flow id
- For the day timestamp we can take the millis since epoch for the end of the day (24:00h).
- columns include runids, start time, end time, snapshot time
- This table will also be used to efficiently retrieve the flows that had an activity in a certain day. That is needed for daily aggregations, but also for several UIs, including a flow-based UI.
- *<working on an example layout>*

The flow for Job A will appear on day 1, 2, 3
The flow for job B will appear on day 2, 3, 4
The flow for job C will appear on day 3
The flow for job D will appear on day 3, 4



rowkey	columnfamily i
cluster! (Long.Max - day4) ! username! job B	runid ! run_id_value1 : version1 stime ! run_id_value1 : epoch_timestamp of submit time etime ! run_id_value1 : epoch_timestamp at end
cluster! (Long.Max - day4) ! username2 ! job D	runid ! run_id_value9 : version9 stime ! run_id_value9 : epoch_timestamp etime ! run_id_value9 : epoch_timestamp
cluster! (Long.Max - day3) ! username! job A	runid ! run_id_value14 : version9 stime ! run_id_value14 : epoch_timestamp etime ! run_id_value14 : epoch_timestamp
cluster! (Long.Max - day3) ! username! job B	runid ! run_id_value5 : version1 stime ! run_id_value5 : epoch_timestamp ctime ! run_id_value5 : epoch_timestamp at snapshot time
cluster! (Long.Max - day3 ! username ! job C	runid ! run_id_value21 : version1 stime ! run_id_value21 : epoch_timestamp etime ! run_id_value21 : epoch_timestamp
cluster! (Long.Max - day3 ! username2 ! job D	runid ! run_id_value24 : version1 stime ! run_id_value24 : epoch_timestamp ctime ! run_id_value24 : epoch_timestamp at snapshot time
cluster! (Long.Max - day2) ! username! job A	runid ! run_id_value33 : version1 stime ! run_id_value33: epoch_timestamp ctime ! run_id_value33 : epoch_timestamp at snapshot time

cluster! (Long.Max - day2) ! username! job B	runid ! run_id_value8 : version1 stime ! run_id_value8 : epoch_timestamp ctime ! run_id_value8 : epoch_timestamp at snapshot time
cluster! (Long.Max - day1) ! username! job A	runid ! run_id_value8 : version1 stime ! run_id_value8 : epoch_timestamp ctime ! run_id_value8 : epoch_timestamp at snapshot time