Publishing Application Data to YARN Timeline Service v.2

1. Data Model

1.1 Identifying timeline entities

YARN timeline service v.2 (YTS) introduces brand new data models for applications. Similar to timeline service v.1, data stored in YTS v.2 is still organized as set of “TimelineEntity”. However, TimelineEntity in YTS v.2 is different to and incompatible with the TimelineEntity class in v.1.

As a general rule, any object model of the v.2 API is in package `org.apache.hadoop.yarn.api.records.timelineservice`, while any v.2 object models are in package `org.apache.hadoop.yarn.api.records.timeline`.

Each TimelineEntity object in YTS v.2 is uniquely identified by a tuple <*cluster_name, user_id, flow_name, flowrun_id, app_id, entity_type, entity_id*>. Among all elements, YARN will assist YTS user applications (“users”) to generate contextual data:

- **cluster_id**: The name of the cluster this TimelineEntity belongs to. Set by the RM’s `yarn.resourcemanager.cluster-id` in yarn-site.xml.
- **user_id**: The id of the user that the TimelineEntity belongs to. Normally, this represents the user who run the flow and/or application.
- **flow_name**: The name of a “flow” this TimelineEntity belongs to. A “flow” represents some high-level cluster workflow. For example, a “distributed_grep” flow may consist of a few MapReduce applications followed by a YARN DistributedShell application. Flow is an abstract concept, but can be “instantiated” with concrete flow runs.
- **flowrun_id**: The id of a concrete flowrun this TimelineEntity belongs to. Note that a TimelineEntity must belong to one concrete flowrun, rather than an abstract flow.
- **application_id**: The id of the application this TimelineEntity belongs to, same as YARN’s application id.

YARN will assign cluster_id, user_id, and application_id for each entity. flow_name and flowrun_id can be set by the user with YARN’s application tags. Optionally, users may also want to record the actual version of a flow. Users can use TimelineUtils to set these three items:

- `TimelineUtils.generateFlowNameTag(String flowName)`: generate an application tag with specified flow name.
- `Timelineutils.generateFlowVersionTag(String flowVersion)`: generate an application tag with specified flow version.

- `TimelineUtils.generateFlowRunIdTag(long flowRunId)`: generate an application tag with specified flow run id.

All three items can be added to the application submission context's tag field, for example:

```
ApplicationSubmissionContext appContext =
app.getApplicationSubmissionContext();
// set the flow context as YARN application tags
Set<String> tags = new HashSet<>();
tags.add(TimelineUtils.generateFlowNameTag("distributed grep"));
tags.add(Timelineutils.generateFlowVersionTag("3df8b0d6100530080d2e0decf9e528e5
7c42a90a"));
tags.add(TimelineUtils.generateFlowRunIdTag(System.currentTimeMillis()));
appContext.setApplicationTags(tags);
```

If the flow context is not specified, defaults are supplied for these attributes:

- **Flow name**: the YARN application name (or the application id if the name is not set)
- **Flow run id**: the application start time in Unix time (milliseconds)
- **Flow version**: "1"

Users need to specify the type and an user-recognizable id for the TimelineEntity, represented by the entity_type and entity_id fields of the tuple.

Timeline entity data can be posted incrementally. Posting a timeline entity with a new data field will incrementally add the data, while posting an entity with an existing field will override the data. The user should NOT expect that overridden values are kept as a "historical version" in the underlying storage, although the default storage system of ATS v.2 does so.

1.2 Holding data in TimelineEntities

TimelineEntity objects have some fields to hold timeline data:

- **events**: A set of TimelineEvents, ordered by the timestamp of the events in descending order. Each event contains one id and a map to store related information and is associated with one timestamp.
- **configs**: A map from a string (config name) to a string (config value) representing all configs associated with the entity. Users can post the whole config or a part of it in the configs field.
- **metrics**: A set of metrics related to this entity. There are two types of metrics: single value metric and time series metric. Right now the timeline service only supports single value metric. Each metric item contains metric name (id), value, and what kind of aggregation operation should be performed in this metric (no-op by default).
- **info**: A map from a string (info key name) to an object (info value) to hold up related information for this entity.

It is also possible to represent relationships between entities. Each entity contains a **relatesToEntities** and **isRelatedToEntities** fields to represent relationships to other entities. Both fields are represented by a map from a string (the name of the relationship) to a timeline entity. In this way relationships among entities can be represented as a DAG.

To simplify programming, there are a few predefined types of timeline entities:

- **HierarchicalTimelineEntity**: represent timeline entities that have hierarchical orders.
    - **ClusterEntity**: representing one YARN cluster.
    - **ApplicationEntity**: representing one YARN application
    - **ApplicationAttemptEntity**: representing one application attempt
    - **ContainerEntity**: representing one YARN container
    - **FlowRunEntity**: representing one specific flowrun
    - **QueueEntity**: representing one YARN queue
- **FlowActivityEntity**: representing one “flow activity”, such as a new flow run is generated, or the flow has been updated.
- **UserEntity**: representing one user of the cluster.

2. Working with TimelineClient

Users can continue to use the TimelineClientAPI to publish per-framework data to the Timeline Service v.2. You only need to instantiate the right type of the client to write to v.2 (and make sure timeline v.2 is enabled on the cluster). The methods on TimelineClient suitable for writing to the Timeline Service v.2 are clearly delineated, and they use the v.2 types as arguments.

Timeline Service v.2 putEntities methods come in 2 varieties: `putEntities` and `putEntitiesAsync`. The former is a blocking operation which should be used for writing more critical data (e.g. lifecycle events). The latter is a nonblocking operation. Note that neither has a return value.

Creating a TimelineClient for v.2 involves passing in the application id to the factory method.

Any puts to the same entity id tuple will point to the same TimelineEntity. The final outcome of a sequence of puts to the same entity is defined by the actual time each put takes effect on the storage system.

3. Examples

YARN DistributedShell acts as a nice example to use timeline service v.2. To start the timeline client:

```
      ...
      if (timelineServiceV2) {
            timelineClient = TimelineClient.createTimelineClient(
                appAttemptID.getApplicationId());
            LOG.info("Timeline service V2 client is enabled");
          }
      ...
```

Then register the timeline client with the AMRMClient so that it can post YARN level data:

```
      amRMClient.registerTimelineClient(timelineClient);
```

One example to post data with the timeline client:

```
    ...
    final org.apache.hadoop.yarn.api.records.timelineservice.TimelineEntity
entity =
        new
org.apache.hadoop.yarn.api.records.timelineservice.TimelineEntity();
    entity.setId(appAttemptID.toString());
    entity.setType(DSEntity.DS_APP_ATTEMPT.toString());
    //entity.setDomainId(domainId);
    entity.addInfo("user", appSubmitterUgi.getShortUserName());
    org.apache.hadoop.yarn.api.records.timelineservice.TimelineEvent event =
        new org.apache.hadoop.yarn.api.records.timelineservice.TimelineEvent();
    event.setId(appEvent.toString());
    event.setTimestamp(System.currentTimeMillis());
    entity.addEvent(event);

    try {
      appSubmitterUgi.doAs(new PrivilegedExceptionAction<Object>() {
        @Override
        public TimelinePutResponse run() throws Exception {
          timelineClient.putEntitiesAsync(entity);
          return null;
        }
      });
    } catch (Exception e) {
      LOG.error("App Attempt "
          + (appEvent.equals(DSEvent.DS_APP_ATTEMPT_START) ? "start" : "end")
          + " event could not be published for "
          + appAttemptID,
          e instanceof UndeclaredThrowableException ? e.getCause() : e);
    }
    ...
```