```
Title: [YARN-1530] Timeline server SpanReceiver for htrace tracing data
By: Billie Rinaldi
```

Motivation

HTrace (now entering the Apache Incubator) is a tracing instrumentation system based on Google's Dapper. Hadoop 2.6.0 introduced a dependency on HTrace prior to HTrace entering the Incubator.

HTrace instrumentation may be an easy way for some applications to start using the Yarn Timeline Service. In particular, HBase and Accumulo already use HTrace, so enabling these applications to start sending data to the Yarn Timeline would simply be a matter of configuring them with the proposed TimelineSpanReceiver. This would not require any additional code in the applications. Applications that do not currently use HTrace may also prefer to use the HTrace API rather than designing and populating TimelineEntities manually. An example of code that would populate timeline data is as follows:

```
SpanReceiverHost.getInstance(configuration);

TraceScope ts = Trace.startSpan("span description", Sampler.ALWAYS);
try {
  ...
  ts.getSpan().addTimelineAnnotation("annotation 1");
  ...
} finally {
  ts.close();                       <-- sends span data to timeline server
}
```

HTrace Span Data

Each trace consists of a tree of spans, where a span has a start time and stop time as well as some other data. Each span has a parent span, except for the span at the root of the tree which is referred to as the *root span* of the trace. The root span is the one that initiates a trace.

Each span has a span description, as well as unique IDs for the span itself, its trace, and its parent span. There is also an ID / name for the process initiating the span. Optional additional data include named time annotations and arbitrary key/value annotations.

Following is a JSON representation of the information contained in one HTrace Span.

```
{
  "start": 123l,  // start timestamp of span
  "stop": 456l,   // stop timestamp of span
  "description":  "span description",
  "spanId":  1l,  // unique id of span
  "traceId": 2l,  // unique id of trace this span belongs to
  "parentId": 3l, // unique id of parent span or root spanId
  "processId": "process name" // classname or other identifier for process
  "kvAnnotations": {
    "key1": "value1",
    "key2": "value2"
  },
  "timelineAnnotations": [
    { "message": "annotation 1", "time": 345l },
    { "message": "annotation 2", "time": 346l }
  ],
}
```

Usage Patterns

These are the usage patterns we anticipate supporting through the Yarn Timeline Server. Each references specific timeline query request types listed in the Requests section.

1. Retrieve a list of the most recent traces showing the number of associated spans, the process name of the root / initiating span, and start and stop times for the trace (see Request 1)
2. Retrieve all information about a specific trace given its traceId (see Request 2)
3. Retrieve most recent span data or data for a given spanId (see Requests 3 and 4)

The first two will be used most often. The third is not likely to be needed, but is easy to support as a by-product of supporting the first two.

Something to decide is whether we should attempt to support additional indexing. This is discussed further in the Indexed Retrievals section below.

Requests

The following timeline requests should be sufficient to satisfy these access patterns. Their goals are to retrieve:

1. TraceId and its associated spanIds by trace start time

   ```
   Request: GET http://<host>:<port>/ws/v1/timeline/HTRACE_TRACE
   ```

2. Span data by traceId

```
Request: GET http://<host>:<port>/ws/v1/timeline/HTRACE_SPAN?
                primaryFilter=HTRACE_TRACE:<traceId>
```

3. Span data by spanId

```
Request: GET http://<host>:<port>/ws/v1/timeline/HTRACE_SPAN/<spanId>
```

4. Span data by span start time

```
Request: GET http://<host>:<port>/ws/v1/timeline/HTRACE_SPAN
```

To clarify, *span data* is all data for a span, while *spanId* and *traceId* are just the ids with no associated data.

Retrieving a traceId *and its associated span data* by trace start time is **not** supported.

Data-indexed retrievals such as spanId, traceId, or span data by process name or annotation are discussed in the next section.

Indexed Retrievals

One question is whether to support any indexed retrievals of tracing data, beyond retrieval by IDs as previously discussed. A useful reference point may be the queries that Zipkin allows to be performed on tracing data it collects. The Zipkin query service can be inspected here. In addition to retrieving data by ID, it provides methods for retrieving IDs by service name (perhaps analogous to process name?), span name (description), and annotation (a key/value annotation or the name of a timeline annotation).

Since these methods only retrieve trace IDs, these would be easier to implement in the timeline server than indexed retrieval of all span data for a trace. We could copy a span's process name, span description, and annotations to the trace entity as primary filters when inserting. This would multiply the storage and number of inserts required of the timeline server, so we should consider carefully whether we want to support this. The request to retrieve traceIds by process name, span description, or annotion would look like the following.

```
Request: GET http://<host>:<port>/ws/v1/timeline/HTRACE_TRACE?
                primaryFilter=<filtername>:<filtervalue>
```

Proposed TimelineEntities for each Span

```
PUT http://<host>:<port>/ws/v1/apptimeline/
{
 "entities": [
  {
    "entity": 1l, // span id                          <-- for requests 3 and 4
    "entitytype": "HTRACE_SPAN",
    "starttime": 123l,
    "events": [
      {
        "timestamp": 123l, // start time
        "eventtype": "start"
      },
      {
        "timestamp": 456l, // stop time
        "eventtype": "stop"
      },
      { // timeline annotation 1
        "timestamp": 345l,
        "eventtype": "annotation 1"
      },
      { // timeline annotation 2
        "timestamp": 346l,
        "eventtype": "annotation 2"
      }
    ],
    "relatedentities": {
      "HTRACE_TRACE": [2l] // trace id                   <-- for request 1
    },
    "primaryfilters" : {
      "HTRACE_TRACE" : [2l] // trace id                  <-- for request 2
    },
    "otherinfo": {
      "description": "span description",
      "spanId":  1l,
      "traceId": 2l,
      "parentId": 3l,
      "processId": "process name",
      "key1": "value1", // kv annotation 1
      "key2": "value2" // kv annotation 2
    }
  },
  {
    "entity": 2l, // trace id                           <-- for request 1
    "entitytype": "HTRACE_TRACE",
```

```
    "events": [
      {
        "timestamp": 123l, // start time of root span
        "eventtype": "start"
      },
      {
        "timestamp": 456l, // stop time of root span
        "eventtype": "stop"
      }
    ],
    "primaryfilters": {                 <-- possible additions for indexing
      "HTRACE_PROCESS": ["process name"],
      "HTRACE_DESCRIPTION": ["span description"],
      "HTRACE_ANNOTATION": [
        "annotation 1", "annotation 2" // timeline annotations
      ],
      "key1": "value1", // kv annotation 1
      "key2": "value2", // kv annotation 2
    }
  }
 ]
}
```