Mapping and launching app-level Timeline aggregators

Li Lu
Drafted 02/19/2015

In this document we focus on the application-level aggregators (referred as “aggregators” in the context). We may want to have node-level or rack-level aggregators in future. Discussions on those aggregators remains as future work.

Basic aggregator mapping for YARN applications

Each YARN application is mapped to two logical Timeline aggregators. An ApplicationMaster contacts one aggregator to post all its application specific data (Application Level Aggregator), and the RM contacts the other aggregator for RM specific data of that application (Application Level Aggregator inside RM). Aggregators are the only interface to post data to the YARN Timeline Service. Multiple levels of aggregators may be possible in future but not discussed here. In the below discussion, we start with a notion of AMs/NMs not making any assumptions on the *physical* locations of the aggregators - an aggregator can sit next to the AM in a separate container, it can side on the local node etc.

Implementation

Aggregators (implemented as *BasicTimelineAggregators*) are organized into aggregator-collections (implemented as *TimelineAggregatorsCollection*). Aggregators in the same aggregator-collection may share web-servers (to receive incoming Timeline writes) and/or storage layer connections (to send out aggregated information to the Timeline storage). In Phase I, each NodeManager *i* has one aggregator-collection *C_i*. *C_i* includes (and only includes) the app-specific aggregators mapped to all ApplicationMasters running on NodeManager *i*. The ResourceManager has one aggregator collection *C_rm*. *C_rm* includes all the app-specific aggregations happening inside the RM. Depending on the mapping policy, the total numbers of app level aggregators may be less than the number of running applications - the mapping is surjective.

Depending on the number of running applications (*N_app*) and the number of nodes (*N_node*) or on the size of individual applications, we may choose different mapping policies (implemented as different aggregator-collections). We have multiple options

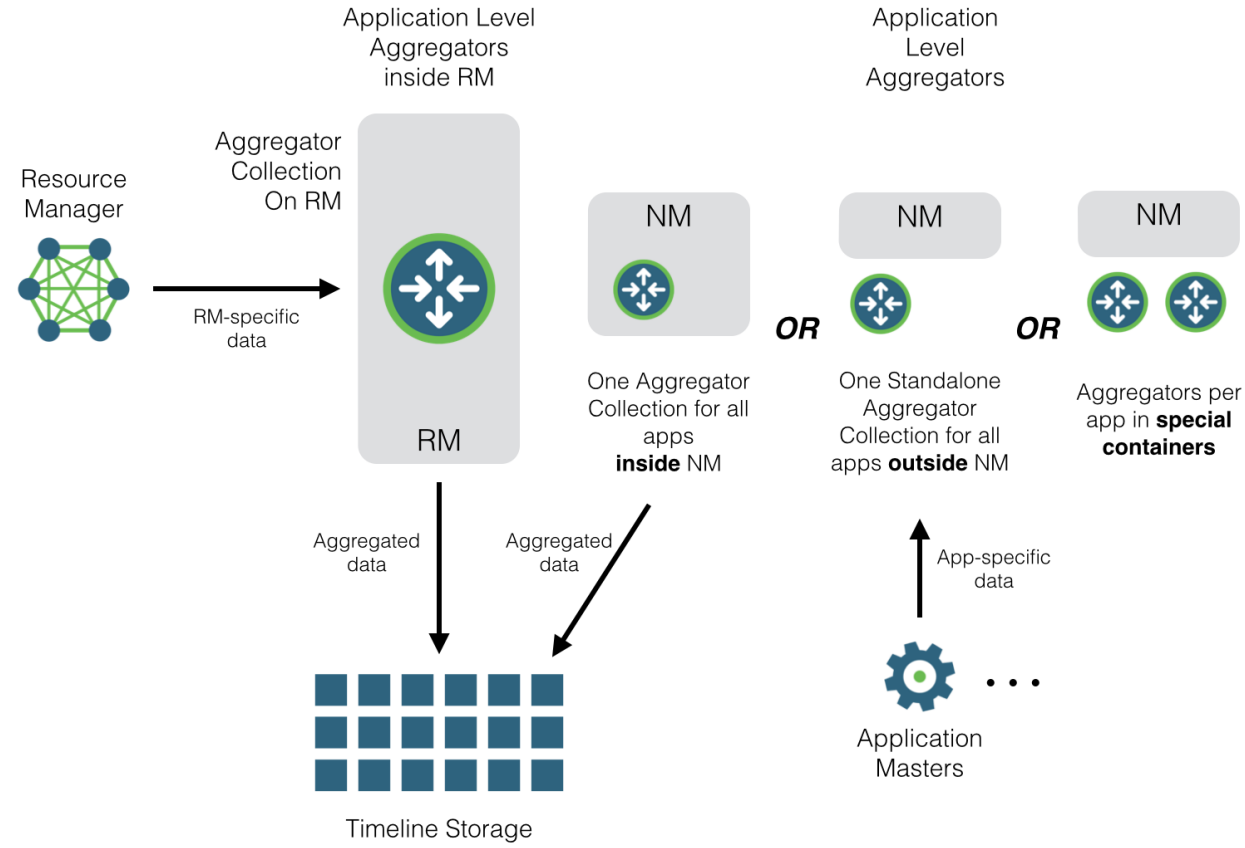
- Case where *N_app* < *N_node*: In this case, each application has a devoted app-level aggregator entity, and all app-level aggregators in RM are mapped to a small number

of aggregator entities in *C_rm*. If the application's AM runs on NodeManager *i*, the aggregator runs in *C_i*.

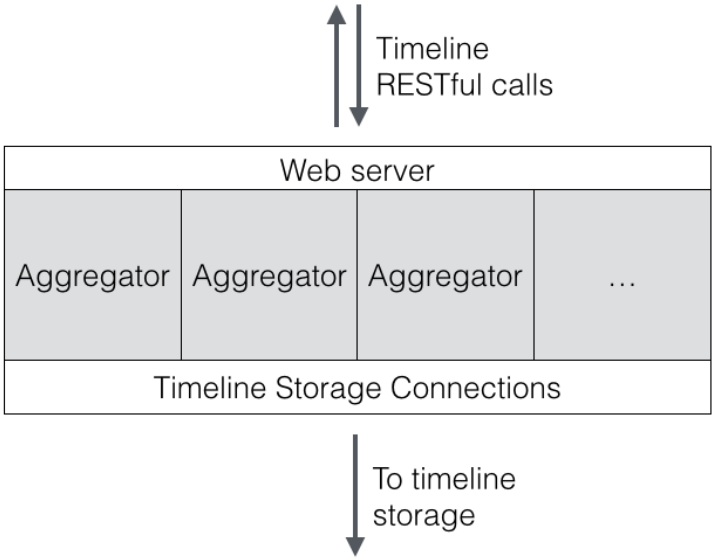
- If *N_app* > *N_node* or *N_app* >> *N_node*, we may consider to launch a constant number of aggregators inside each NodeManager, so the total aggregator entities is bounded by the number of NMs. The reason we'd like to avoid running too many aggregators is the pressure on the storage - too many writers writing to say HBase RegionServers. We can override the aggregator mapping in this case.

- The other dimension to this choice is the size of an application. For a very large application, keeping the aggregator inside the NodeManager is going to be an isolation-challenge. So, we should try to spawn a separate aggregator container for large applications.

System overview



Aggregator collections



Launching aggregator collections

Launching aggregator collections on NMs (YARN-3033)

- As auxiliary service (YARN-3030, mostly done): PerNodeAggregatorServer implemented as a NodeManager auxiliary service. Work remained: verify if we can launch aggregator collections in this way and YARN-3087
- As a standalone service (the rest part of YARN-3033): implement a new service that performs the same task as PerNodeAggregatorServer, but runs as a standalone service. Create configurations to switch between auxiliary service mode and standalone mode. Let's do this a little later after the POC.

Launching aggregator collections on RMs (YARN-3034)

- Launching an aggregator collection inside RM.