YARN–321 : The Generic Application History Service

- Last date of modification: Dec 19 2013
- By: Vinod Kumar Vavilapalli, Zhijie Shen, Mayank Bansal

Problem statement

YARN solves the generic resource-management problem in a generic fashion. But today, there is no general solution for recording history of applications that are run in YARN. ResourceManager has a small cache of completed applications, but once that is forgotten (after limits are hit or if RM crashed), all history about the finished applications is lost.

State of the art

The MapReduce specific JobHistoryServer (JHS) is an implementation that is custom made for MapReduce - the first and largest application on top of YARN thus far. At present, it needs to be deployed as a separate trusted server alongside YARN. A MapReduce JobHistoryServer reads JobHistory files written by individual MapReduce jobs (MR ApplicationMasters) and so has to coevolve with the MapReduce runtime. By extension, a version of MapReduce framework has to work with the corresponding JobHistoryServer of the same version - any changes to the format of the underlying JobHistoryFiles forces an upgrade of both the MapReduce framework as well as the JobHistoryServer.

Beyond MapReduce, alternate frameworks on top of YARN including the sample Distributed Shell, Apache Tez, Giraph on YARN etc don’t have a generic solution for storing their application history. Today they are left to fend off for themselves by implementing a custom solution and/or a trusted HistoryServer specific to that framework - a not so scalable solution. If there is no custom server, they can only depend on the small cache in RM’s memory till the application is flushed out of the cache.

Proposal overview

Solve the application-history problem in a generic fashion with a generic

ApplicationHistoryServer (AHS). There are two parts to it:

- Remember generic information about completed apps. Generic information includes
    - Application level data like queue name, user information etc in the ApplicationSubmissionContext
    - List of ApplicationAttempts run for the app
    - Information about each ApplicationAttempt
    - List of containers run under each ApplicationAttempt and
    - Generic information about each container.
- Remember per-framework information about completed apps.
    - The per-framework information is completely specific to the framework in question
    - For e.g, for MapReduce framework, this includes things like number of map tasks, reduce tasks, counters etc.

> This document only proposes a solution for storing the generic information about applications. Solution for the per-framework information is tracked in a separate JIRA.

Design

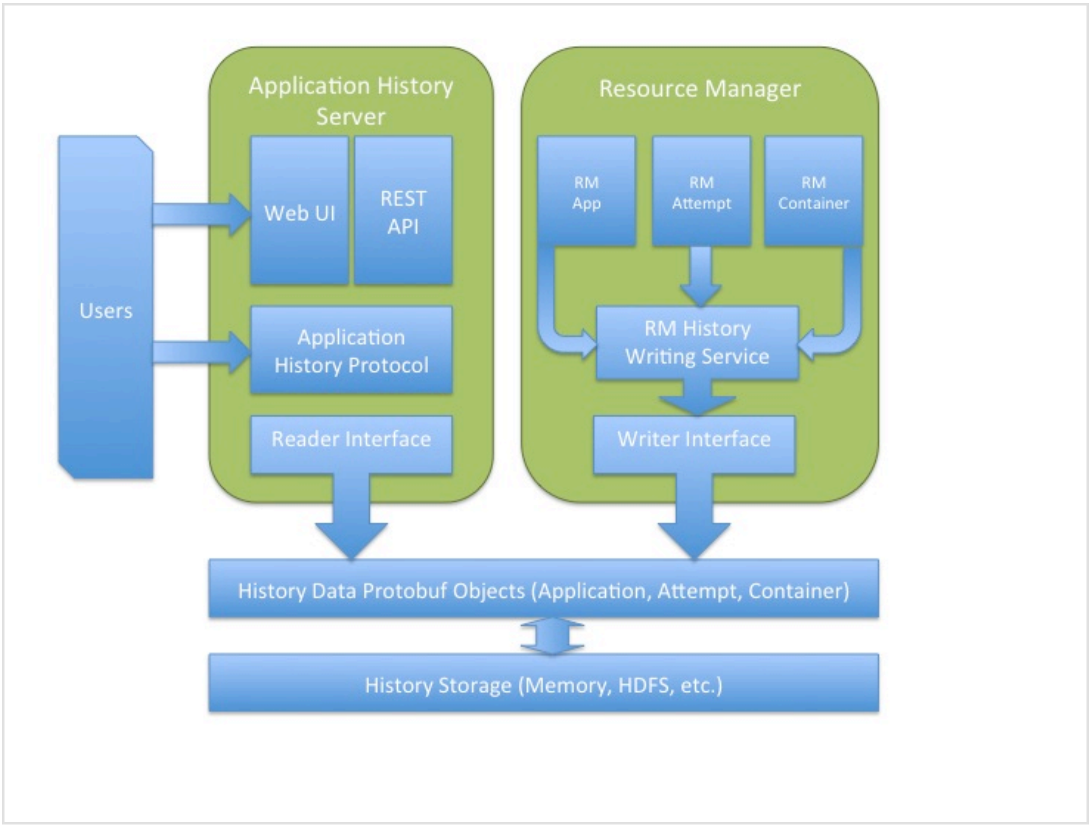
Basics

- The ResourceManager will write per-application data to a (very likely) thin and pluggable HistoryStorage layer.
- The ResourceManager will push the data to HistoryStorage using a separate thread which is flushed after an application finishes.
- ApplicationHistory uses a reader HistoryStorage interface to read and serve generic application data.

HistoryStorage

- HistoryStorage is different from the current RMStateStore (where ResourceManager's state is stored for recovery) and so, unlike JobHistory files, HistoryStorage isn't used for state-tracking or as a transaction log.
- ResourceManager will try to publish information about completed apps in a best-case manner but there will always be a few edge conditions during restart of ResourceManager where it is not possible to guarantee the data storage.
- HistoryStorage will have APIs for publishing app-info, retrieving appinfo and listing completed apps.
- HistoryStorage may have different implementations
    - A file based implementation where RM writes per-app files to HDFS, which will take care of file-management like we do today in JobHistoryServer and serve users by reading the data in files
    - A shared bus implementation where RM directly writes to AHS and AHS persists them in a storage that it controls Files/DB etc.
    - Or an implementation where the shared-bus is a database.
- To start with, we will have an implementation with a per-app HDFS file.

Here's a representation of the components and how they are placed:


Application History Server
Resource Manager
Users
Web UI
REST API
RM App
RM Attempt
RM Container
Application History Protocol
RM History Writing Service
Reader Interface
Writer Interface
History Data Protobuf Objects (Application, Attempt, Container)
History Storage (Memory, HDFS, etc.)

Generic Application History Service

ApplicationHistoryServer

- *Running as process*: ApplicationHistoryServer will be a separate process, which is independent of ResourceManager, such that it doesn't need to be synchronous with the life cycle of ResourceManager and deployment is more flexible.
- *Caching*: Any caching of information should be transparent to the underlying HistoryStorage.
- *Filtering*: A number of filters should be provided to efficiently retrieve a subset of the historic application collection in HistoryStorage.
- *Aggregated logs*: Logs will be served and potentially log management (expiry etc.) by ApplicationHistoryService via an abstract LogService component.
- *Security*: ApplicationHistoryService will have security from start, using tokens similar to how the JHS does.

Miscellaneous

- *User interfaces*: Command line clients and/or webclients will have RPC, web and REST interfaces to interact with ApplicationHistoryService to get info about finished applications. Fundamentally, we'll have two types of interfaces
  - Per-app info. And more granular related APIs for getting AppAttempt and container specific information
  - List of all apps
    - Querying list of apps based on username, queuename etc. To start with, we can either imitate what ResourceManager does in REST interface only. Ultimately, we'd like to make RPC and web have the consistent filtering capability. In addition, it may be good to push the filtering computation to HistoryStorage for better performance.