Compacted Aggregated Logs

YARN-2942

1/15/15 -- Robert Kanter

Problem

Turning on log aggregation allows users to easily store container logs in HDFS and subsequently view them in the YARN web UIs from a central place. Currently, there is a separate log file for each Node Manager. This can be a problem for HDFS if you have a cluster with many nodes as you'll slowly start accumulating many (possibly small) files per YARN application. The current "solution" for this problem is to configure YARN (actually the JHS) to automatically delete these files after some amount of time.

Proposal

We can provide a better solution by compacting the aggregated log files for each node into a single log file per application.

The per-node aggregated log files currently reside all in the same directory, per application and are in a binary format (`AggregatedLogFormat`, which is essentially a `TFile`). Each file consists of some header metadata (ACLs, owner, and version), followed by key-value pairs. Each of the keys is a Container ID and the values are the logs (stdout, stderr, and syslog are all here) of that container and their lengths.

The end goal is to have a procedure (described later) which will combine these aggregated log files into a single compacted aggregated log file. The format of this file can be similar to the aggregated log files; we can essentially just "dump" the information from the aggregated log file into it, which gives us the advantage of reusing most of the log parsing code that already exists. However, the JHS currently finds the logs for a specific Container ID in an aggregated log file by searching through it -- this is somewhat inefficient and would be even worse for the larger compacted file. The solution to this is to have a second file which stores the index, which can simply be a mapping of the container IDs with their offsets and lengths in the compacted log file; we can also put the header metadata in here as well. Here's what the format of the index file would roughly look like:

```
|-----------------------------|
|Version                      |
|ACLs                         |
|Owner                        |
|ContainerID,offset,length    |
|...                          |
|-----------------------------|
```

The original design was to have the ResourceManager trigger an arbitrary NodeManager to do the compaction of all of the aggregated logs for a particular Application. This has some

failure scenarios that would have to be addressed (e.g. what if that NM goes down, etc); however, the main problem with this approach is that there isn't currently a practical way to know when all nodes used by an Application are done uploading their aggregated logs.

This has brought about the need for a more distributed approach. Each NodeManager will append it's aggregated logs to the compacted aggregated log file after uploading the aggregated log to HDFS. ZooKeeper/Curator will be used to store one lock per Application to ensure that only one NodeManager can append to the compacted aggregated log at a time. Once an aggregated log file has been appended, we can delete it. In short, after uploading it's aggregated log file, each NodeManager would essentially do the following:

1. Acquire lock
2. Does my aggregated log file exist?
    a. Yes → append it then delete it
3. Release lock

Because of this bottleneck at compacting, we'll still keep the aggregated log behavior so that the user can get logs sooner. The JHS (and presumably eventually ATS) code for retrieving logs will be updated to check the compacted aggregated log's index file for a Container ID and only fallback to the aggregated log if it can't find that ID. This will ensure that the user can get all of the logs as soon as they are aggregated, while they're being compacted, and after they're compacted.

The NodeManager will write the Container ID to the index, then append the log for that container, and then write the offset and length for that container to the index. If an NM dies while appending, the next NM can recognize that the previous NM didn't finish because the offset and/or length will be missing in the index. This will allow a subsequent NM, or a JHS/ATS reader such as the JHS, to recognize that the logs for that container are incomplete. In the case of the NM writer, it can "fix" the index and append after it. In the case of the JHS/ATS reader, it can ignore that index entry and trigger the fallback behavior to serve the missing container's logs from the original aggregated log file (it won't have been deleted).

We can include some configuration options, such as:

- enable/disable compaction
    - (would automatically be disabled if log aggregation is disabled)
- location for compacted log files
- how long to retain compacted log files
    - This would require a cleanup service
- ZooKeeper/Curator configs

Follow-up Work

- Aggregated logs from long running services (which are uploaded on a per container basis instead of per node) are not considered here at this time (perhaps YARN-2548

can update it to handle them). For now, they should be left alone by the compacting code.

- A cleanup service for compacted logs
- As explained above, if an NM dies while appending its aggregated log file, the currently being written container's logs and subsequent containers' logs from that node will not be in the compacted file. While this is okay, it would be better to somehow recover from this. Perhaps the next NM can complete the append for it. Or if the NM restarts, it can check if it has an aggregated log file that needs to be appended.