YARN log servlet improvements

Introduction

Reading logs

There are two fundamental ways how to read logs from YARN applications

- You can read them directly from the storage where it's stored (HDFS, AWS, Azure etc.)
- Using any tool that is based on the LogAggregationFileController's readAggregatedLogs method (Java applications)

The second method is recommended because:

- The aggregation format and folder structures are subject to change
- The aggregated logs are not necessarily human-readable. IndexedFile format contains binary pieces, that are not human-readable.

LogAggregationFileController call hierarchy

Tools already using the LogAggregationFileController:

- Mapreduce CLI's logs command
- YARN logs CLI
- NMWebServices#getLogs()
    - Only works for running containers (used by the YARN logs CLI) in that NodeManager
- LogWebServiceUtils#getStreamingOutput()
    - Utility that produces a streaming output. Used by:
    - ATSv1.5 in AHSWebServices#getContainerLogFile
        - Bound in AHSWebApp#setup(), which is called from ApplicationHistoryServer#startWebApp
    - ATSv2 in LogWebService#getContainerLogFile
        - Bound in TimelineReaderServer#startTimelineReaderWebApp

The existing code in JobHistory Server

The JobHistory Server currently only supports the HTML rendering of aggregated logs.

- The /logs path is bound in HsWebApp.java.
    - Call chain: HsController#logs() -> render(HsLogsPage.class) -> AggregatedLogsBlock#render() -> fileController#renderAggregatedLogsBlock()
    - So ultimately the LogAggregationFileController renders this as well (same Java code)

Suggested enhancements

Pull out common code pieces

ATSv1.5 and ATSv2 has lots of common code that can be pulled to an abstract service / package. The logic is the same, the code is *almost* the same.
The only ATS specific thing in that AppInfo is constructed from an ApplicationReport, which information is extracted from the TimelineReader client. Later appInfo's user and appState are used (see JHS part), but this is the only dependency from the timeline part.

Add ability for ATS to read logs of running apps

Currently neither version of the AHS is able to read logs of running apps (local logs of NodeManager). YARN log CLI is integrated with NodeManager to extract local logs as well.
Call chain:

- NMWebServices#getContainerLogFile
- Called through REST by the /containers/{containerid}/logs/{filename} endpoint in LogsCLI#getResponeFromNMWebService
- Called in LogsCLI#printContainerLogsFromRunningApplication ...

Since in the previous step we've extracted the common piece of code into an abstract class, we can add this functionality to the servlet.
Context: YARN-5224

Integrate the new abstract service to JHS

In HsWebApp#setup() a new bind should be added, which is the JHS implementation of the abstract webapp (isolated in the previous step).
ATS must be substituted to collect ApplicationReport: it should be requested from the RM using a RMClient (already in JHS). This may negligibly increase RM-JHS load.

Add option to UIv2 to get container logs from the new JHS API

Provided the new API is ready to use, we can add a new configuration option to UIv2 to decide which endpoint should the log request target.

Remove code duplicates from utility classes

Also, build a centralized package for handling everything regarding (preferably in yarn-common).
There is tons of code duplication in the following classes

- ContainerLogsUtils
- LogWebServiceUtils
- LogToolUtils
- LogAggregationWebUtils
- LogCLIHelpers

General purpose log request with additional query parameters

The current endpoints are robust but not very flexible with regards to filtering options. I suggest to add an endpoint which provides filtering options.
E.g.:
In ATS we have multiple endpoints:
`/containers/{containerid}/logs/{filename}`
`/containerlogs/{containerid}/{filename}`
We could add @QueryParams to the rest endpoints like this:
`/containers/{containerid}/logs?fileName=stderr&containerState=FAILED&nodeId=nm45`
Thus we could add some extra filtering options.

New feature: regex

To demonstrate the new capabilities and how easy it will be to add a functionality to all log servlets at the same time: add the ability to search in the aggregated logs with a given regex.
A conceptual use case:

- Users run several MR jobs daily, but some of them fail to localize a particular resource at first. We want to search these YARN applications, and extract their logs/find what is common in them (probably some failed underlying disk in a particular node).