Improve YARN shared cache support for LinuxContainerExecutor

Zhenzhao Wang

Problem Statement 1
High-Level Approach 2
Relevant Background or Prior Art 2
Prior Art 2
Problems And Potential Improvements 3
Goals and Non-Goals 3

Components **3**
Resource Visibility Semantic In MR 3
Resource/Staging Dir Structure 4
Shared Cache Semantic 6
Visibility Setting Vs Shared Cache Setting 7
Public Shared Cache Only 8
Non-Public Shared Cache 8
Backward Compatibility 8

Alternatives Considered **9**
Keeping Dir Structure And Changing Permission Only 9

Overview

Problem Statement

Resources' visibility and upload policy are controlled by two sets of flags in yarn now. Shared Cache Uploader will try to upload resource to shared cache regardless of resource's visibility if upload policy is true. And current MR job will set upload policy to true while setting resources' visibility to non-public. It presents a problem while running LinuxContainerExecutor in a secure mode (yarn.nodemanager.linux-container-executor.nonsecure-mode.limit-users set to false).

High-Level Approach

Provide visibility flags that user could specify for resources to MR clients.
Provide yarn aligned shared cache policy for MR clients.

Relevant Background or Prior Art

Prior Art

According to current code, the problem will be triggered if people's enabled shared cache for resources by default. Here are more details:

There are four kinds of resources in MR: jobjar, libjar, files, archive. Libjars, files and archive share a more similar process while the jobjar's behavior is little different.

- Upload Policy.
  The upload policy of resource is controlled by two flags.

Flag Name	Description
yarn.sharedcache.enabled	Enable shared cache
mapreduce.job.sharedcache.mode	A comma delimited list of resource categories to submit to the shared cache. The valid categories are: jobjar, libjars, files, archives. If "disabled" is specified then the job submission code will not use the shared cache.

If both flags are set, the NM will try to upload data.

- Resource visibility.
  The behavior of job jar is different from libjar, files and archives here.
    - JobJar's visibility is controlled by "mapreduce.job.jobjar.visibility" which is false by default. Note that this is a flag which could be modified by MR framework.

mapreduce.job.jobjar.visibility	Resource Visibility
False	Application
True	Public

The "mapreduce.job.jobjar.visibility" flag will be overridden and set to true when shared cache is enabled and the jar is found in shared cache.

    - libjars , libfiles and libarchives's visibility is controlled resource's ACL. If a resource is world-readable then it's visibility is Public, otherwise it will be private resource. By

default the resource file permission is 644 which means public in yarn. Actually, it's hard to set a resource to be private from MR.

Problems And Potential Improvements

- Note that the visibility and shared cache policy exposed to MR user are different from yarn's shared cached and visibility semantic now. Thus we could see follow problems:
    - SC Uploader will try to upload data while upload policy is set true even if resource is non-public. It's non viable while LCE is enabled. And the request itself might introduce a permission escalation. From yarn's perspective, user are asking data to be uploaded to public dir even when they want the data to be non-public.
    - The setting and behavior of visibility is vague for MR user.
        - For job jar, 1. if the job jar will be overridden to public when we found jar in shared cache even if the we set job jar to non visible. 2. If shared cache is enabled, data will be uploaded to shared cache and make it world readable even if we set jar to non visible.
        - For libjar, libarchives and libfiles, the visibility is actually decided by the default permission of creating files in HDFS. If LCE is enabled, we couldn't use shared cache for such resources as it will always be set to private and shared cache uploader couldn't upload it shared cache (more confirmation needed with non-local resource).
- From yarn's perspective, a resource will be set to world readable if it's uploaded to shared cache dir now. We should think about whether to provide user-based/public or ACL based shared cache.

Goals and Non-Goals

1. Support LCE in shared cache manager.
2. Provide clearer configure semantics for MR jobs.

Components

Resource Visibility Semantic In MR

The first problem we might need to discuss here is whether we need to provide visibility settings in MR framework. I guess it will be more clear to users if we provide explicit settings, especially for non job jar resources. Here're the semantics which are going to provide.

- ***PUBLIC***: Shared by all users.
- ***PRIVATE***: Shared among all applications of the same user.
- ***APPLICATION:*** Shared only among containers of the same application.

Note the semantic here is consistent with what we have on node manager of yarn. Keeping the semantic consistent has several benefits:

1. It is good to prevent permission escalation.
2. It make the permission model more clear. It could help avoid permission visibility confusion, chaos from up to down.

The new setting format will be like:

Type	Prior Format	New Format
jobjar	Mapreduce.job.jobjar.visibility :{ture, false} It's generated flag	mapreduce.job.jobjar.visibility :{public, private, application} Default value will be application
files	-files $file1,$file2. It could only be private on HDFS as the staging dir is private, but possible to be uploaded to shared cache.	-files ${file1}#{public,private, application},${file2}#{public,private, application}
libjars	-libjars $jar1,$jar2 It could only be private on HDFS as the staging dir is private, but possible to be uploaded to shared cache.	-libjars ${jar1}#{public,private, application},${jar2}#{public,private, application} Default will be application
archives	-archives $arch1,$arch2 It could only be private on HDFS as the staging dir is private, but possible to be uploaded to shared cache.	-archives ${arch1}#{public,private, application},${arch2}#{public, private, application} Default will be application

Resource/Staging Dir Structure

Current job staging dir structure is
${yarn.app.mapreduce.am.staging-dir}

```
|  -- $username
        | -- .staging  rwx------
              | -- $jobID rwx------
                    | -- files rwx------
                          | … rw-r--r--
                    | -- libjars rwx------
```

| … *rw-r--r--*
| -- archives *rwx------*
| … *rw-r--r--*
| -- job.jar *rw-r--r--*
| -- job.xml *rw-r--r--*
| -- log4j *rw-r--r--*
| -- job.splitmetainfo *rw-r--r--*
| -- job.split *rw-r--r--*

The new structure will be

Current job staging dir structure is
${yarn.app.mapreduce.am.staging-dir}
| -- $username
| -- .staging *rwx------*
| -- $jobID *rwx-----x*
| -- application *rwx-----*
| -- job.xml *rw-r--r--*
| -- log4j *rw-r--r--*
| -- job.splitmetainfo *rw-r--r--*
| -- job.split *rw-r--r--*
| -- files *rwx------*
| … *rw-r--r--*
| -- libjars *rwx------*
| … *rw-r--r--*
| -- archives *rwx------*
| … *rw-r--r--*
| -- private *rwx-----*
| -- files *rwx------*
| … *rw-r--r--*
| -- libjars *rwx------*
| … *rw-r--r--*
| -- archives *rwx------*
| … *rw-r--r--*
| -- public *rwxr-xr-x*
| -- files *rwx------*
| … *rw-r--r--*
| -- libjars *rwx------*
| … *rw-r--r--*
| -- archives *rwx------*
| … *rw-r--r--*
| -- job.jar (based on configuration)

Shared Cache Semantic

Shared cached resource semantic:

- ***PUBLIC***: Shared by all users.
- ***PRIVATE***: Shared among all applications of the same user.
- ***APPLICATION:*** Shared only among containers of the same application
- ***None***: Disable shared cache for the resource

Note that we will only upload shared cache when it's public resource, private and application will be probably be supported in the future when we got non public shared cache support.

Resource Type	Prior Art		New format	
	User facing flag Yarn.sharedcache.enabled=true is the precondition	Yarn facing info	User facing	Yarn facing
Job jar	Mapreduce.job.sharedcache.mode contains jobjar	Auto generated resource type. Application while not found in shared cache. Public while found in shared cache. Node manager will try to upload the resource regardless resource type.	User could specify resource type via libjar, files and archives	User specified resource type. Node manager will only try to upload public resource now.
libjars	Mapreduce.job.sharedcache.mode contains libjars	Auto generated resource type. Would always be private as the staging dir is private. Node manager will try to upload	User could specify resource type via libjar, files and archives	User specified resource type. Node manager will only try to upload public resource now.

		the resource regardless resource type.		
archives	Mapreduce.job.sharedcache.mode contains archives	Auto generated resource type. Would always be private as the staging dir is private	User could specify resource type via libjar, files and archives	User specified resource type. Node manager will only try to upload public resource now.
files	Mapreduce.job.sharedcache.mode contains files	Auto generated resource type. Would always be private as the staging dir is private. Node manager will try to upload the resource regardless resource type.	User could specify resource type via libjar, files and archives	User specified resource type. Node manager will only try to upload public resource now.

Visibility Setting Vs Shared Cache Setting

With the old approach, the visibility setting might be conflicted with shared cache settings. For example, a non visibile might be set to upload to world readable shared cache. In such cases, MR/yarn need to decide whether to keep resource's visibility or keep shared cache settings work. MR used to choose to upload resource regardless it's application/private type. With the new proposed approach, the visibility semantic is consistent with yarn resource type. And yarn will only try to upload public resource unless nonpublic shared cache is support in the future. Here's the table of possible conflicts:

Potential Conflict type	Action
Resource is application/private, however the resource might be already in shared cache, should we use the data from shared cache then?	Skip shared cache check for the moment as the shared cache is world readable

Resource is application/private, however shared cache enabled is also true.	We should avoid upload data to shared cache.

Public Shared Cache Only

The data in shared cache dir is in 555 mode and not configurable now. And the shared cache uploader need resource to have a public visibility while LCE is enabled. So one simple solution is that we only support shared cache for public resources.

Non-Public Shared Cache

In this patch, we will consider public shared cache only.
If we want to support non public shared cache, we have to think about following problems.

- What kind of semantic we want to provide? Should it be ACL based or private/group/public based?
- How is directory structured?
- How could the uploader read local resource if data is non public? The approach might vary based on the semantic provided.

For ACL based solution, here're the details. (TBD)
For private/public based solution, here're the details. (TBD)

Backward Compatibility

This table assumes cluster use 2.9 version. And we tested the compatibility with 2.6 clients.

User Client Version	Map reduce Job	Share cache
2.6 client without any hadoop jar bundled	Good	Good
2.6 Client bundled every hadoop jar	Good	Works if the jar is already in shared cache. Otherwise share cache wouldn't work
2.6 Client bundled every hadoop jar && add hadoop 2.6 jar in libjar && HADOOP_USER_CLASSPATH_FIRST=True && mapreduce.job.user.classpath=true	BAD due to Message missing required fields: update_request	BAD

	s[0].container_id, update_requests[0].update_type	
2.6 Client bundle part of hadoop jar. E.g. bundled commons but not hadoop-map-reduce-client-core	Undetermined	Undetermined
2.9 client and 2.6 servers	? Might be bad due to 2.9 client is using a different dir structure	? Might be bad due to 2.9 client is using a different dir structure

Alternatives Considered

Keeping Dir Structure And Changing Permission Only

We could keep dir structure unchanged and just change the files/dirs's permission. It will be a quick solution. However, it will bring more confusion to developers and users in the long run.