Extending the YARN resource model for easier resource-type management and profiles

Varun Vasudev with suggestions from Bikas Saha, Wangda Tan, Hitesh Shah, Jian He, Siddharth Seth, Vinod Kumar Vavilapalli, Sidharta Seethana

Last modified: July 12, 2015

Overview

Currently, there are efforts to add support for various resource-types such as disk(YARN-2139), network(YARN-2140), and HDFS bandwidth(YARN-2681). This document is a proposal to extend the YARN resource model to a more flexible model which makes it easier to add new countable resource-types. It also considers the related aspect of “resource profiles” which allow users to easily specify the resources they need for containers.
When it comes to adding support for a new resource-type, there are two aspects to be considered - scheduling and isolation. This document deals only with the scheduling aspect.

Background

Today, adding a new countable resource-type in YARN is a fairly involved process. A lot of work has to be carried out to modify the `ProtocolBuffers` file(s) as well as the corresponding Java source code. That has to be followed up by modifying the `DominantResourceCalculator` class to add support for the new resource-type. In addition, numerous unrelated test cases have to be modified since they use the

`Resource.newInstance(memory, vcores)` function. Further, most new resource-types are treated the same - they're just named differently. All in all, today, 37 source files have to be modified to add support for a new disk resource-type; and a similar(probably greater) will have to be modified for future resource-types and most of the changes will look the same as earlier changes, with the only difference being the name of the variable.

We propose changing the resource model to allow support for arbitrary, countable resources which can be defined by cluster administrators. When we speak of countable resources, we refer to resource-types where the allocation and release of resources is a simple subtraction and addition operation. We don't wish to cover exclusive resources such as ports or non-countable resources such as labels in this version of the document. However there is nothing in our proposal that precludes adding support for resource-types such as labels and such work can be carried out as part of a future effort.

Extending the YARN resource model

ResourceManager

In our proposed model, we add a new configuration file - "`resource-types.xml`" which can be used by administrators to add new resource-types. "`resource-types.xml`" is used by the ResourceManager(RM) to determine the set of resource-types for which scheduling is enabled. These configurations can be set in `yarn-site.xml` as well, but it might be cleaner to specify them in a separate file. The resources configuration file must contain memory and vcores as resource-types to prevent any loss of functionality. Administrators can add arbitrary new resource-types, as long as the resources they represent are countable.

A sample `resource-types.xml` file is below:

```
<configuration>
   <property>
     <name>yarn.scheduler.resource-types</name>
     <value>memory,cpu</value>
   </property>
   <property>
     <name>yarn.scheduler.resource-types.memory.name</name>
     <value>yarn.io/memory</value>
   </property>
   <property>
     <name>yarn.scheduler.resource-types.memory.units</name>
     <value>MB</value>
   </property>
```

```
    <property>
      <name>yarn.scheduler.resource-types.memory.type</name>
      <value>countable</value>
    </property>
    <property>
      <name>yarn.scheduler.resource-types.memory.enabled</name>
      <value>true</value>
    </property>
    <property>
      <name>yarn.scheduler.resource-types.cpu.name</name>
      <value>yarn.io/cpu</value>
    </property>
    <property>
      <name>yarn.scheduler.resource-types.cpu.units</name>
      <value>vcores</value>
    </property>
    <property>
      <name>yarn.scheduler.resource-types.cpu.type</name>
      <value>countable</value>
    </property>
    <property>
      <name>yarn.scheduler.resource-types.cpu.enabled</name>
      <value>true</value>
    </property>
<configuration>
```

“yarn.io/memory” and “yarn.io/cpu” map to the existing memory and vcores resource-types that YARN currently supports. The resource name is similar to the one from the Kubernetes resource model(a link can be found in the references section).

NodeManager

Similarly, we propose adding a new file `“node-resources.xml”` where the resource capabilities of a node can be specified. Like `“resource-types.xml”`, these configurations can be set in `yarn-site.xml` as well, but it might be cleaner to specify them in a separate file.

A sample node-resources.xml is below:

```
<configuration>
  <property>
      <name>yarn.nodemanager.resource-types</name>
      <value>memory,cpu</value>
    </property>
  <property>
      <name>yarn.nodemanager.resource-types.memory.name</name>
      <value>yarn.io/memory</value>
    </property>
    <property>
      <name>yarn.nodemanager.resource-types.memory.units</name>
      <value>MB</value>
    </property>
    <property>
      <name>yarn.nodemanager.resource-types.memory.type</name>
      <value>countable</value>
    </property>
    <property>
      <name>yarn.nodemanager.resource-types.memory.available</name>
      <value>32</value>
    <property>
      <name>yarn.nodemanager.resource-types.cpu</name>
      <value>yarn.io/cpu</value>
    </property>
    <property>
      <name>yarn.nodemanager.resource-types.cpu.units</name>
      <value>vcores</value>
    </property>
    <property>
      <name>yarn.nodemanager.resource-types.cpu.type</name>
      <value>countable</value>
    </property>
    <property>
      <name>yarn.nodemanager.resource-types.cpu.available</name>
      <value>8</value>
    </property>
<configuration>
```

The `NodeStatusUpdaterImpl` class will be modified to report values for all the resources specified in the `"node-resources.xml"` file instead of just memory and cpu like it currently does.

Resource class and DominantResourceCalculator

We propose adding a new function to the YARN resource model:

```
public static Resource newInstance(Map<ResourceTypeInformation, Long>
resources)
```

and deprecating

```
public static Resource newInstance(int memory, int vcores)
```

The definition of the ResourceTypeInformation class would be:

```
public class ResourceTypeInformation {
    URI name;
    String units;
    ResourceTypes type;
    Boolean enabled;
}
```

where ResourceTypes is an enum of the possible resource-types(“countable” being the only type currently supported). An additional restriction is that the `“name”` field must be unique. The name field will act as the identifier. This is to avoid situations where two resource-types have the same name but different units or types which would lead to confusion. The `“enabled”` field is to allow administrators an easy way to enable or disable a resource-type. Currently, the `“units” field` has two purposes:

1. it will be used during NM registration to ensure that the RM and NM are using the same units.
2. documentation

The corresponding change to the `ProtocolBuffers` definition would be:

```
enum ResourceTypes {
  COUNTABLE = 0;
}

message ResourceMapEntry {
  string key = 1;
  int64 value = 2;
  string units = 3;
  ResourceTypes type = 4;
}

message ResourceProto {
 optional int32 memory = 1;
 optional int32 virtual_cores = 2;
 optional repeated ResourceMapEntry resource_value_map = 3;
}
```

When the NM registers with the RM, if there is a mismatch in the resource-types listed in `"resource-types.xml"` and the resources reported by the NM(missing resource-types, additional resource-types, mismatched units, mismatched type, etc) an exception is thrown causing the NM to shut down. It should be noted that `"node-resources.xml"` does not have the `"enabled"` field. This is because it should be of no concern to the NM if a resource-type is enabled or not.

The existing memory and cpu resource-types will be mapped to “yarn.io/memory” and “yarn.io/cpu” respectively. To preserve compatibility and ensure that there’s no loss of functionality, “yarn.io/memory” and “yarn.io/cpu” will be designated as mandatory resource-types. The decision to use URIs to serve as the name of the resource-types is derived from Kubernetes which proposes using a RFC 1123 compliant domain followed by a “/” followed by a name.

Existing usage of the `Resource.newInstance(int memory, int vcores)` function will be changed to switch to the `Resource.newInstance(Map<ResourceTypeInformation, Long>)` method. The existing `getMemory(), setMemory(int memory), getVirtualCores(), setVirtualCores(int vCores)` functions will continue to work without any change in behaviour. Internally, these calls will be mapped to read/write the “yarn.io/memory” and “yarn.io/cpu” elements. The `hashCode` function in the Resource class will have to be modified to take into account all the resource-types.

The `DominantResourceCalculator` will change to carry out calculations for every resource-type that has the enabled flag set in `Map<ResourceTypeInformation, Long>` as opposed to the current named variables.

NodeStatusUpdaterImpl

Today the NMs report resources to the RM in the `NodeStatusUpdaterImpl`. Once we allow arbitrary resource-types, the `NodeStatusUpdaterImpl` will have to read the `"node-resources.xml"` file and report the resource values set in that file.

In addition, there might be scenarios where the value of the resource-type requires access to node resources(such as the /proc filesystem). To allow for this, we will need to modify the NM to allow plugins to read and process such information. For now, we don’t intend to add support for this. However, if there is a strong sentiment that support for this must be added, we are open to it.

Adding or removing resource-types

Due to the new configuration files and the way the proposed system is structured, ordering of operations is important when adding or removing resource-types.

When adding a new resource-type, the NMs must be upgraded first followed by the RM. This allows the NMs to register with the RM without leading to a mismatch.

Conversely, when removing a resource-type, the RM must be upgraded first(to remove the resource-type) followed by the NMs.

RegisterApplicationMasterResponse

Currently, we have a field in the `RegisterApplicationMasterResponse` - `scheduler_resource_types` - which can be used by applications to determine the resource-types being used by the RM for scheduling. However, this field is currently an enum. We propose deprecating this field and adding a new field "`scheduler_resource_types_info`" which will contain all the information for the resource-types that are enabled for scheduling. The changes proposed will be as follows:

```
message RegisterApplicationMasterResponseProto {
  optional ResourceProto maximumCapability = 1;
  optional bytes client_to_am_token_master_key = 2;
  repeated ApplicationACLMapProto application_ACLs = 3;
  repeated ContainerProto containers_from_previous_attempts = 4;
  optional string queue = 5;
  repeated NMTokenProto nm_tokens_from_previous_attempts = 6;
  // scheduer_resource_types is deprecated
  repeated SchedulerResourceTypes scheduler_resource_types = 7;
  repeated ResourceMapEntry sceduler_resource_types_info = 8;
}
```

and add the following APIs to the `RegisterApplicationMasterResponse` class:

```
public abstract Map<URI, ResourceTypeInformation>
getSchedulerResourceTypesInformation();

public abstract void setSchedulerResourceTypesInformation(Map<URI,
ResourceTypeInformation>);
```

We propose deprecating -

```
public abstract EnumSet<SchedulerResourceTypes>
getSchedulerResourceTypes();
```
and
```
public abstract void
setSchedulerResourceTypes(EnumSet<SchedulerResourceTypes> types);
```

Resource Profiles

As YARN adds support for additional resource-types, specifying and configuring resource allocations for containers becomes more and more cumbersome. We propose resource-profiles as a solution to allow users submitting jobs to easily specify the resources they need.

Today, for each resource-type, admins have to specify the minimum and maximum allocation in yarn-site.xml, which ends up leading to a whole bunch of minimum and maximum allocation properties in `yarn-site.xml`.

In the case of memory and vcores, the minimum allocation also functions as the step when doing calculations for allocations. While this is applicable for memory and vcores, it may not apply to resource-types in the future.

In addition, the minimum allocation also functions as the default allocation - which may be appropriate for memory and vcores but may not be ideal for other resource-types.

From an end-user’s perspective, the current model means specifying every resource that the scheduler knows about, especially if the minimum allocation does not satisfy your requirement - which could lead to asking for excess resources(in cases where the user does not know the exact amount they require). This becomes especially cumbersome if the scheduler adds new resource-types(e.g. a memory scheduled cluster that turns on cpu scheduling) - now the user has to figure out the right value for vCores for his job, which was running fine so far.

Resource profiles are a proposal to address this issue.

Essentially, we allow cluster admins to setup a config file(“resource-profiles.xml”) which lets them create shorthands for certain resource profiles. Again, like “resource-types.xml” and “node-resources.xml” these profiles can be specified as a part of “yarn-site.xml”. An example of such a file is

```
<configuration>
   <property>
     <name>yarn.scheduler.profiles</name>
     <value>pf1,pf2,pf3,pf4,pf5,pf6</value>
   </property>
   <property>
     <name>yarn.scheduler.profile.pf1.name</name>
     <value>minimum</value>
   </property>
   <!-- set value for memory -->
   <property>
     <name>yarn.scheduler.profile.pf1.yarn.io/memory</name>
     <value>1024</value>
   </property>
   <!-- set value for cpu -->
   <property>
     <name>yarn.scheduler.profile.pf1.yarn.io/cpu</name>
     <value>1</value>
   </property>
   <property>
     <name>yarn.scheduler.profile.pf2.name</name>
     <value>maximum</value>
   </property>
   <!-- set value for memory -->
   <property>
     <name>yarn.scheduler.profile.pf2.yarn.io/memory</name>
     <value>8192</value>
   </property>
   <!-- set value for cpu -->
   <property>
     <name>yarn.scheduler.profile.pf2.yarn.io/cpu</name>
     <value>8</value>
   </property>
   <property>
     <name>yarn.scheduler.profile.pf3.name</name>
     <value>default</value>
   </property>
```

```
<!-- set value for memory -->
<property>
  <name>yarn.scheduler.profile.pf3.yarn.io/memory</name>
  <value>2048</value>
</property>
<!-- set value for cpu -->
<property>
  <name>yarn.scheduler.profile.pf3.yarn.io/cpu</name>
  <value>2</value>
</property>
<property>
  <name>yarn.scheduler.profile.pf4.name</name>
  <value>small</value>
</property>
<!-- set value for memory -->
<property>
  <name>yarn.scheduler.profile.pf4.yarn.io/memory</name>
  <value>1024</value>
</property>
<!-- set value for cpu -->
<property>
  <name>yarn.scheduler.profile.pf4.yarn.io/cpu</name>
  <value>1</value>
</property>
<property>
  <name>yarn.scheduler.profile.pf5.name</name>
  <value>medium</value>
</property>
<!-- set value for memory -->
<property>
  <name>yarn.scheduler.profile.pf5.yarn.io/memory</name>
  <value>3072</value>
</property>
<!-- set value for cpu -->
<property>
  <name>yarn.scheduler.profile.pf5.yarn.io/cpu</name>
  <value>3</value>
</property>
<property>
  <name>yarn.scheduler.profile.pf6.name</name>
  <value>large</value>
</property>
<!-- set value for memory -->
```

```
   <property>
     <name>yarn.scheduler.profile.pf6.yarn.io/memory</name>
     <value>8192</value>
   </property>
   <!-- set value for cpu -->
   <property>
     <name>yarn.scheduler.profile.pf6.yarn.io/cpu</name>
     <value>8</value>
   </property>
</configuration>
```

The file format above is a little awkward. This is mainly due to the restrictions of the YarnConfiguration class. Alternate file formats which are easier to understand are discussed below in the “Configuration File Formats” section.

This example sets up 3 profiles - “small”, “medium”, and “large”. “minimum”, “maximum” and “default” are profiles reserved by YARN to allow admins to specify the minimum, maximum and default allocations for each resource-type.

When a user submits a job, instead of specifying each resource-type, he can specify a profile name instead. If no profile is mentioned, the “default” profile is used.

The benefits of this model is that going forward, if admins decide to enable new resource-types, users won’t need to update all their jobs - admins should be able to configure better allocations given the profile. It also makes it easier for users to request for resources; they (hopefully) will end up making better decisions on allocations since they are used to specifying profile names when deploying instances on various cloud platforms. This becomes especially useful as we add support for network and disk as resource-types, where it might be harder for users to specify exact needs.

This model ends up collecting all resource allocations related configurations into a single file making it easier to manage.

If users specify allocations for a memory and/or cpu in addition to a profile, then that is as seen as an override. In case of the profiles example detailed above, and using distributed shell as an example application, “—container_memory 3072 —container_profile small” indicates that you wish for an allocation of a container with 3072MB and 1 vCore. This also implies that existing jobs which specify memory and/or vcores should see no change in the allocation of memory and/or vcores. This override behaviour will only be allowed for memory and cpu to preserve existing behaviour. Specifically, we propose that if an app does the following:

```
ResourceRequest remoteRequest = ResourceRequest.newInstance(priority,
  hostname, capability, numContainers);
Map<URI, ResourceMapEntry> overrides = new HashMap<>();
ResourceMapEntry value =
  ResourceMapEntry.newInstance("yarn.io/memory", 16 * 1024, "MB",
  ResourceType.COUNTABLE);
overrides.put(new URI("yarn.io/memory", entry));
Resource profileCapabilitiesOverrides =
  Resource.newInstance(overrides);
// targetProfile is the name of the profile
ProfileCapability capability =
ProfileCapability.newInstance(targetProfile,
  profileCapabilitiesOverrides);
remoteRequest.setProfileCapability(capability);
```

then, the final capability assigned by the RM will be the resources specified in the profile except that the memory allocated will be 16GB. However, if an app cannot use both the capabilities and the profiles model. As mentioned above, the overrides will only be allowed for memory and cpu. We also propose a config flag to enable/disable overrides(set to enable overrides by default) using which admins can turn off the override behaviour.

The changes to YARN are primarily to the `ResourceRequestProto -`

```
message ProfileCapabilityProto {
     string profile = 1;
     ResourceProto profileCapabilityOverride = 2;
}

message ResourceRequestProto {
  optional PriorityProto priority = 1;
  optional string resource_name = 2;
  optional ResourceProto capability = 3;
  optional int32 num_containers = 4;
  optional bool relax_locality = 5 [default = true];
  optional string node_label_expression = 6;
  ProfileCapabilityProto string profile = 7;
}
```

We also propose a change to the RegisterApplicationMasterResponse:

```
message RegisterApplicationMasterResponseProto {
  optional ResourceProto maximumCapability = 1;
  optional bytes client_to_am_token_master_key = 2;
  repeated ApplicationACLMapProto application_ACLs = 3;
  repeated ContainerProto containers_from_previous_attempts = 4;
  optional string queue = 5;
  repeated NMTokenProto nm_tokens_from_previous_attempts = 6;
  repeated SchedulerResourceTypes scheduler_resource_types = 7;
  optional ResourceProfilesProto resource_profiles = 8;
}
```

ResourceProfilesProto is defined below in the “Resource profiles API” section.

When a new ask comes in, the `ApplicationMasterService` looks up the registered profiles, and uses the settings in the profile to set the final ask values. This is done in the `allocate(AllocateRequest request)` function itself. Before the sanity check is carried out, the `capability` is set in the `ResourceRequest` to take into account the profile specified. As a result, we don’t foresee changes to other aspects of the scheduler code.

A similar change will have to be carried out in the `ClientRMService` to allow profiles to be used when launching `ApplicationMasters`.

Resource profiles API

To allow YARN clients and application masters to know the list of profiles available, we need to provide an API in the RM. The proposal it to add the following APIs -

```
Protocol Buffers
message ResourceProfileEntry {
  string name = 1;
  repeated ResourceProto resources = 2;
}
```

```
message ResourceProfilesProto {
 repeated ResourceProfileEntry resource_profiles_map = 3;
}


RPC
class GetResourceProfilesRequest {
  // empty class
}

class GetResourceProfilesResponse {
  public Map<String, Map<URI, Long>> getAllResourceProfiles();
  public Map<URI, Long> getResourceProfile(String profile);
}
GetProfilesResponse getResourceProfiles(GetProfilesRequest request)

REST
GET /resource-profiles
```

Configuration File Formats

The sample files described above are in the format currently used by the YarnConfiguration class. However, all the files can probably be expressed in a more concise manner. The concise forms are below. If there is interest in the community to switch to the more concise forms, we can use them instead.

```
"resource-types.xml"

<configuration>
   <resource>
     <name>yarn.io/memory</name>
     <units>MB</units>
     <type>countable</type>
     <enabled>true</enabled>
   </resource>
   <resource>
     <name>yarn.io/cpu</name>
     <units>vcores</units>
     <type>countable</type>
     <enabled>true</enabled>
   </resource>
<configuration>
```

“node-resources.xml”

```
<configuration>
  <resource>
    <name>yarn.io/memory</name>
    <value>8192</value>
    <units>MB</units>
    <type>countable</type>
  </resource>
  <resource>
    <name>yarn.io/cpu</name>
    <units>vcores</units>
    <type>countable</type>
    <value>8</value>
  </resource>
<configuration>
```

“resource-profiles.xml”

```
<configuration>
   <profiles>
     <profile>
       <name>minimum</name>
       <resource>
         <name>yarn.io/memory</name>
         <value>1024</value>
       </resource>
       <resource>
         <name>yarn.io/cpu</name>
         <value>1</value>
       </resource>
     </profile>
     <profile>
       <name>maximum</name>
       <resource>
         <name>yarn.io/memory</name>
         <value>8192</value>
       </resource>
       <resource>
         <name>yarn.io/cpu</name>
         <value>8</value>
       </resource>
```

```
</profile>
<profile>
  <name>default</name>
  <resource>
    <name>yarn.io/memory</name>
    <value>2048</value>
  </resource>
  <resource>
    <name>yarn.io/cpu</name>
    <value>2</value>
  </resource>
</profile>
<profile>
  <name>small</name>
  <resource>
    <name>yarn.io/memory</name>
    <value>1024</value>
  </resource>
  <resource>
    <name>yarn.io/cpu</name>
    <value>1</value>
  </resource>
</profile>
<profile>
  <name>medium</name>
  <resource>
    <name>yarn.io/memory</name>
    <value>3072</value>
  </resource>
  <resource>
    <name>yarn.io/cpu</name>
    <value>3</value>
  </resource>
</profile>
<profile>
  <name>large</name>
  <resource>
    <name>yarn.io/memory</name>
    <value>8192</value>
  </resource>
  <resource>
    <name>yarn.io/cpu</name>
    <value>8</value>
```

```
      </resource>
    </profile>
  </profiles>
<configuration>

“resource-profiles.json”
{
  “minimum”: {
    “yarn.io/memory”: 1024,
    “yarn.io/cpu”: 1
  },
  “maximum”: {
    “yarn.io/memory”: 8192,
    “yarn.io/cpu”: 8
  },
  “default”: {
    “yarn.io/memory”: 2048,
    “yarn.io/cpu”: 2
  },
  “small”: {
   “yarn.io/memory”: 1024,
   “yarn.io/cpu”: 1
  },
  “medium”: {
    “yarn.io/memory”: 3072,
    “yarn.io/cpu”: 3
  },
  “large”: {
    “yarn.io/memory”: 8192,
    “yarn.io/cpu”: 8
  }
}
```

References

1. YARN-2139 - https://issues.apache.org/jira/browse/YARN-2139
2. YARN-2140 - https://issues.apache.org/jira/browse/YARN-2140
3. YARN-2681 - https://issues.apache.org/jira/browse/YARN-2681

4. Kubernetes resource model - https://github.com/GoogleCloudPlatform/kubernetes/blob/master/docs/design/resources.md