Add FPGA resource provider for YARN-3926

Motivation

The resource model of Hadoop YARN will go through refactoring as described in YARN-3926 in order to extend the YARN resource model to a more flexible model which makes it easier to add new countable resource types like CPU, memory and HDFS bandwidth. However, current YARN-3926 design does not consider exclusive resource like GPU, FPGA, network ports, etc.

Current gaps of YARN-3926 for supporting FPGA is as follows:

1. The scheduler only considers non-exclusive resource. The exclusive resources may have extra attributes needs to be matched when scheduling. Not just simply add or reduce a number. For instance, in our PoC, a FPGA slot in one node may already have one IP flashed so that the scheduler should try to match this IP attribute to reuse it.
2. No NM side resource management of FPGA resource. For instance, dynamically resource discovery, monitoring and preparation before container launch. This is vendor specific and we should have a plugin framework for different vendor to implement. The interfaces may consists "listDevice, monitorDeviceHealth".
3. Device resource needs additional preparation and isolation before container launch. For instance, FPGA device may need to download an IP file from a repo then flash to an allocated FPGA slot. This is also vendor specific and should be pluggable. We can try extend the ResourceHandlerModule introduced by YARN-3366 to achieve this.

4. YARN-3926 is introducing “profiles” which are predefined amounts of resources that the user can choose from. The initial design seems doesn’t support client API to override the exclusive resource amounts.

The new YARN resource model is ideal if it could be general and efficient for managing both exclusive and non-exclusive resource to serve different kinds of workloads. Enhancing YARN-3926 will be a yearlong process so we need to approach the ideal implementation of the resource model step-by-step.

A practical way maybe we split it into two steps.
First, keep current YARN-3926 scheduler feature unchanged, then finish the NM side local resource scheduler and plugin framework design to implement a suboptimal solution. This new component will track node's exclusive resources and connect the container to the real device.
Secondly, remove NM side local resource scheduler and try unify all types of resource in a global scheduler. Here we propose the first solution details in the following sections.

Basic Goals

- The users are able to config static FPGA devices in the node that YARN can use
- Use resource profile (YARN-3926) to support specifying FPGA as a resource for RM to schedule to a certain node
- In NM, support local FPGA resource scheduler to assign/cleanup N FPGA slots to/from a container
- Support vendor plugin framework with basic features
    - Plugin code is packaged and compiled with code base (should be changed in the future) at first
    - Plugin have interfaces that can meet basic vendor requirements

Stretch Goals

v1.5

- Extend vendor plugin framework with advanced feature
    - Plugin version management
    - Lifecycle management
    - Lazy load
    - Automatically discover FPGA devices
    - Long-running vendor plugin
- Support Docker runtime for FPGA applications

V2.0

- Monitoring of FPGA devices
- Isolation of different type of FPGA devices

Solution overview

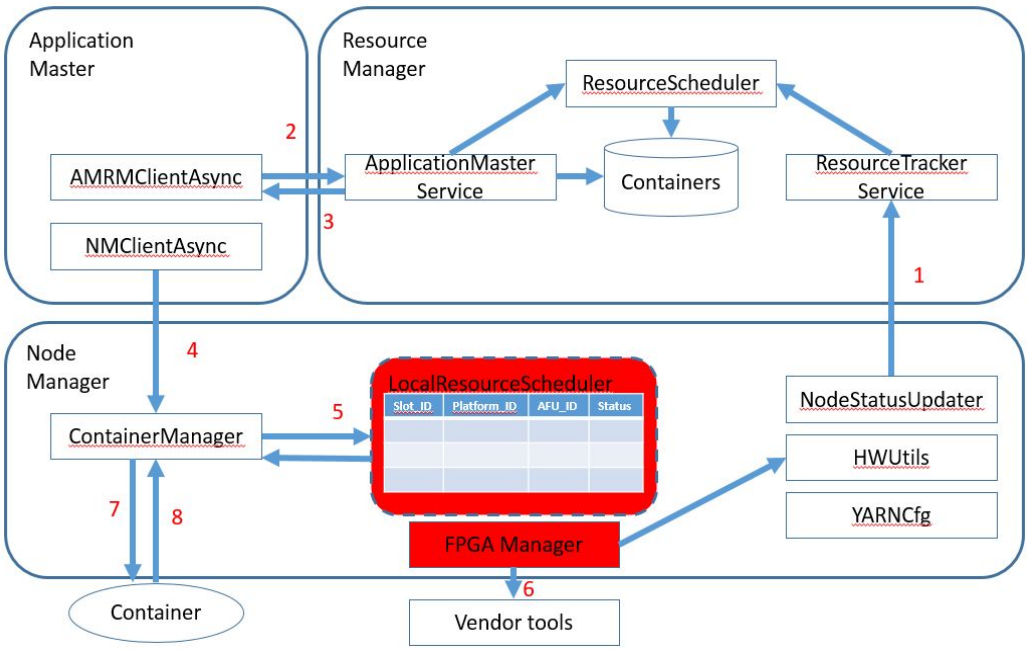


Figure 1. Solution architecture

To achieve the basic goals, the key is adding a local resource scheduler to NM which is responsible for allocating or cleaning up the FPGA devices. And YARN already has a resource handler abstraction(YARN-3443) that can be utilized to implement the container hooks to configure FPGA devices.

A typical workflow is as follows:

1. The admin config FPGA devices that NM can use in YARN configuration (resource-type.xml and node-resources.xml for defining FPGA resource and its count. node-resources.xml(need modification) for specifying different types of allowed FPGA device minor numbers). NM will initialize local FPGA resource scheduler with allowed FPGA devices and also report the resources to RM.
2. Application master requests container with a resource profile containing N FPGA devices(may have different type). Currently the container just consists numeric FPGA resource but no detailed device information.
3. RM allocate the containers.
4. AM set the IP UUID/name in container environment and sends requests to NM to launch the allocated containers.
5. Prior to each container launch, NM will ask FPGA local scheduler for which FPGA devices can be allocated and use vendor specific plugin to configure the device for container.
6. If a FPGA slots needs re-configuration, NM will use vendor plugin to reconfigure or isolate the FPGA slots if needed.
7. Launch the container.

8. After the container complete, NM will inform the local FPGA resource scheduler to clean up the FPGA resources.

Detailed Design

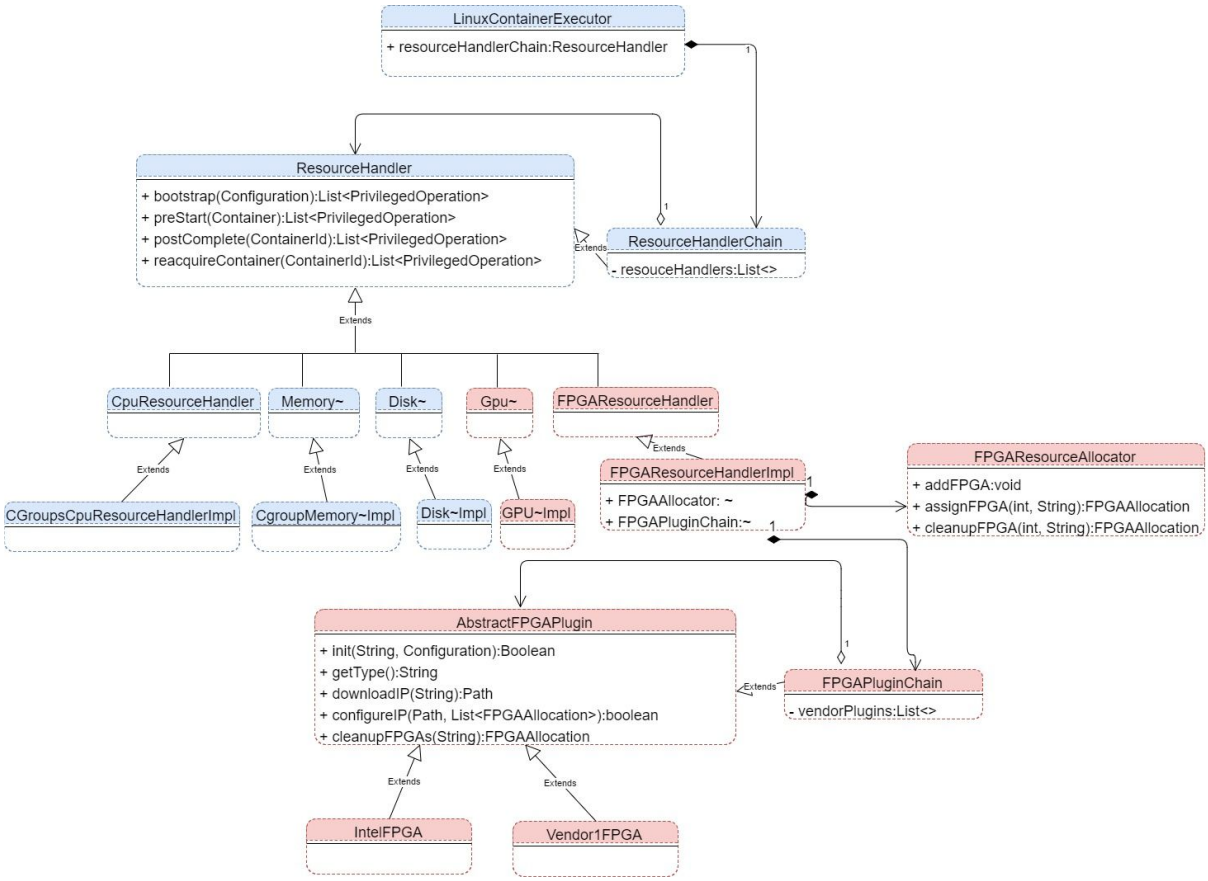


Figure2. Basic design UML

We'll add a "FPGAResourceAllocator" and implement a "FPGAResourceHandlerImpl" for the container hooks. And it consists a FPGA plugin chain to handle different type of FPGA vendor specific requirements.
In FPGAResourceHandlerImpl, the container hooks will be implemented as follows:

- The bootstrap method will invoke each plugin's init method(validate the system environment) and handle adding all different type of real FPGA devices to "FPGAResourceAllocator"
- The preStart method will request FPGAResourceAllocator to assign FPGA slots for a container based on the resource type. If needed, it will download and configure IP.
- The postComplete will request FPGAResourceAllocator to recycle unused FPGA slots from a finished/failed container for each type of FPGA resource

Implementation

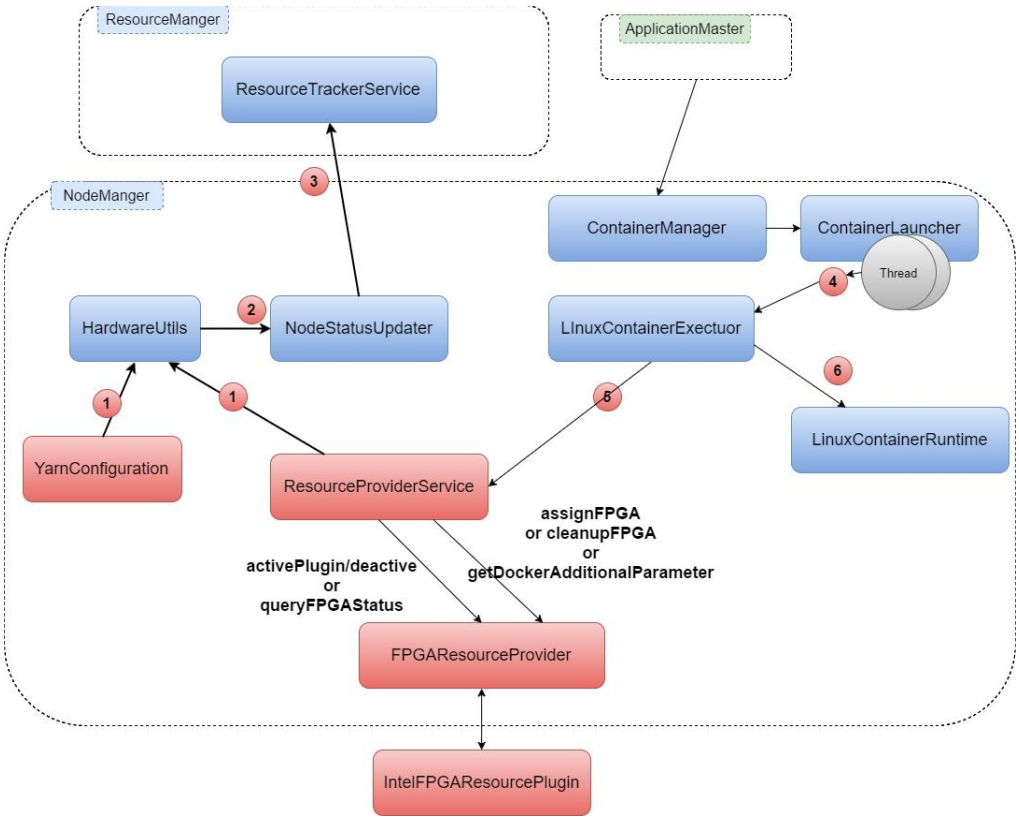
The modifications consists below parts:

1. Static properties in node-resource.xml for allowed device number of certain type
2. A new class FPGAResourceHandler
3. A new class FPGAResourceAllocator
4. A simple plugin interface and vendor's implementation

Open Discussion

1. Stretch Goals' design

A draft design of the stretch goals is as follows. The basic idea is each vendor plugin is a long running instance on each NM and providing interfaces for NM (ResourceProviderService) to interact with.



The ResourceProviderService consists of all exclusive resource provider and delegate below requests to specific resource provider:

- activePlugin/deactivePlug
- queryFPGAStatus
- assignFPGA/cleanupFPGA
- getDockerAdditionalParameter

When a FPGA type requests comes to FPGAResourceProvider, it will delegate to proper vendor plugin to handle it. In this case, the vendor plugin becomes a long-running service that does all the above dirty works.
First of all, the admin should install FPGA vendor specific packages (including kernel mode driver and user model driver) and install YARN FPGA vendor specific resource plugin(long running) and configure it into YARN.

A typical event flow:

A. When start YARN:
    a. The NM activates the enabled resource provider based on configuration.
    b. NM hardwareUtils will report FPGA resource to RM after query plugin.
B. Before launch container, FPGA resource handler will use plugin to:
    a. Assign FPGA
    b. Cleanup FPGA
    c. Isolate FPGA is handled by existing cgroup device handler
    d. Get additional Docker additional parameter and set to container's context environment
C. When launch Docker container, LCE runtime should check the Docker related environment and then build the whole run command.