Implementation notes for supporting FPGA as a resource in YARN

Tangzhankun

1. FPGA first-class citizen support motivation

As the dark silicon challenge comes and the AI workloads requirements grows, we've seen more and more accelerator hardwares involved in both big data and machine learning area. In essence, new algorithms and methodologies are created utilizing dedicated hardware for solving various computation difficulties.

And FPGA is one of the big player in these area. Both research and industry have plenty examples of using FPGA in big data processing like compression/decompression/encryption, cloud provider's product(MS Azure and Bing) and AI inference phase.
Comparing with GPU, FPGA is shown to be valuable for its higher flexibility, lower cost and lower latency.

Category[1]	Examples[2]	Reference[3]	Performance Gain[4]
Dense Linear Algebra	Gaussian Elimination, K-means	K-means	8.4X
Sparse Linear Algebra	Finite Element Analysis, PDE, Circuit Simulation	Circuit Simulation	5X
Spectral Methods	FFT	FFT	5.7x
N-Body Methods	Molecular Dynamics	Molecular Dynamics	168x
Structured Grids	Image Processing, Physics	Particle Grid	15.6x
Unstructured Grids	Computational Fluid Dynamics	Unstructured Search	20X
Map Reduce	Distributed Searching, Monte Carlo Simulations	Monte-Carlo Simulations	31x
Combinational Logic	CRC, Checksums, AES, Hashing,	RNG	25x
Graph Traversal	Search, Sort	Traversal Cache	70x
Dynamic Programming	Genome string matching	Smith-Waterman	10x-50x
Graphical Models	Neural Networks, HMM, Viterbi	CNN (Intel PEG)	10X
Backtracking / B&B	Integer Linear Programming	Backtracking DNA Sequencing	20X
Finite State Machines	Video codecs, Data Mining, Compression	GZIP Compression (IBM)	12x

1. Berkeley dwarfs that capture the pattern of computation and communication for different workloads. See http://view.eecs.berkeley.edu/wiki/Dwarfs
2. Typical application areas for a particular dwarf
3. Link to reference implementation of an example workload
4. Speedup of FPGA over Xeon (assuming technology and resource scaling of performance is linear and applies similarly to both FGPA and CPU)

The Landscape of Parallel Computing Research: A View from Berkeleyhttp://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf

To fulfill the potential acceleration requirements on YARN, we propose supporting FPGA as a first class citizen. With this, the big data and AI application can request FPGA resource easily and evolve with the “App+Accelerator” trend.

2. Design and implementation

Firstly, I’ll list the goals in two phases:

- Phase1 (Done)
  - Discover one type of vendor FPGA devices
  - Allocate FPGA devices with local scheduler in NM
  - Isolate vendor FPGA devices with cgroups (native FPGA module)
  - Launch FPGA application with simple container under cgroup
  - Reacquire container after NM restart
  - One default vendor plugin
- Phase2
  - Node constraints update after FPGA devices reprogramming
  - Docker support for container launching with FPGA
  - TBD

From the end-user’s point of view, to enable FPGA devices in YARN:
The root user admin should:

1. Configure container-executor.cfg like GPU native module to specify allowed FPGA devices

```
[fpga]
# Enable / disable the module
module.enabled=true
# Major device number of FPGA
```

```
fpga.major-device-number=247
# Allowed minor device numbers, empty means all FPGA devices managed by YARN.
fpga.allowed-device-minor-numbers=0,1,2
```

2. Configure yarn-site.xml to specify 4 configurations. The resource plugin, vendor plugin, plugin required executable path and allowed FPGA devices' subset

```
 <property>
  <description>
   Enable additional discovery/isolation of resources on the NodeManager,
   split by comma. By default, this is empty.
   Acceptable values: { "yarn-io/gpu", "yarn-io/fpga"}.
  </description>
  <name>yarn.nodemanager.resource-plugins</name>
  <value>yarn.io/fpga</value>
 </property>
 <property>
  <description>
   When yarn.nodemanager.resource.fpga.allowed-fpga-devices=auto specified,
   YARN NodeManager needs to run FPGA discovery binary (now only support
   Intel's aocl) to get FPGA information.
   When value is empty (default), YARN NodeManager will try to locate
   discovery executable from system environment "ALTERAOCLSDKROOT"
  </description>
  <name>yarn.nodemanager.resource-plugins.fpga.path-to-discovery-executables</name>
  <value>/home/fpga/intelFPGA_pro/17.0/hld/bin/aocl</value>
 </property>
 <property>
  <description>
   Specify FPGA devices which can be managed by YARN NodeManager, split by comma
   Number of FPGA devices will be reported to RM to make scheduling decisions.
   Set to auto (default) let YARN automatically discover FPGA resource from
   system.
   Manually specify FPGA devices if admin only want subset of FPGA devices managed by YARN.
   At present, since we can only configure one major number in c-e.cfg, FPGA device is
   identified by their minor device number. A common approach to get minor
   device number of FPGA is using "aocl diagnose" and check uevent with device name.
  </description>
  <name>yarn.nodemanager.resource-plugins.fpga.allowed-fpga-devices</name>
  <value>0,1</value>
 </property>
 <property>
  <name>yarn.nodemanager.resource-plugins.fpga.vendor-plugin.class</name>
<value>org.apache.hadoop.yarn.server.nodemanager.containermanager.resourceplugin.fpga.IntelFpgaOpenclPlugin
</value>
 </property>
```

After configured FPGA in YARN, the application developer should:

Use YARN-3926 interfaces to request certain amount FPGA devices:

```
setResourceValue(ResourceInformation.FPGA_URI, <count>)
```

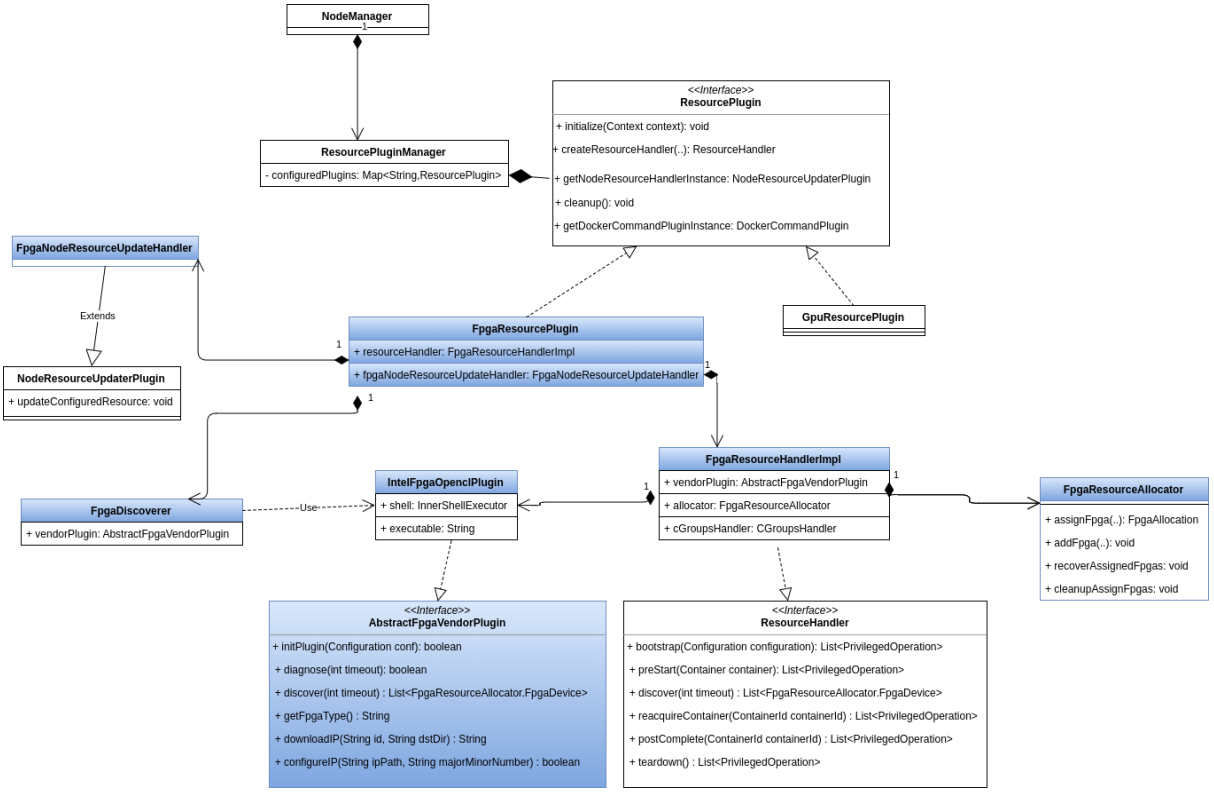
And maybe

```
shellEnv.put("REQUESTED_FPGA_IP_ID", <IP file ID>)
```

As we all know, the FPGA devices should be re-programmed before container launch with correct IP file so that can the application invoke native library/runtime to work.
For now, we only provides developer one environment variable “REQUESTED_FPGA_IP_ID” as a hint for reprogramming stuff. If set, the value of it should be one string like “matrix_mul” indicating an ID of the IP file. The vendor plugins should understand it, download the IP file based on it and reprogram all devices with this same IP after that as needed. If not set, we will assume the devices already with correct IP (might be scheduled by node constraint, or the application knows the IP file location and can load it at runtime) and won’t do device configuration.

To be more clear, if a container requests 2 FPGAs and set the environment variable value “matrix_mul”, the default “IntelFpgaOpenclPlugin” will find “.aocx” files with “matrix_mul” in its name from the container sandbox directory (uploaded to HDFS by client). And then reprogram the 2 FPGAs with this “aocx” file using its tool chain.

At present, we’ve archived the phase 1 goals, the detailed implementation will be illustrated next. In general, the FPGA design follows the already merged GPU architecture design. As the figure below shows:



It has a “FpgaResourcePlugin” which redirect the resource discover thing to “FpgaDiscoverer”, the allocation/isolation things to “FpgaResourceHandlerImpl”. And

these things are mainly done through IntelFpgaOpenclPlugin and “FpgaResourceAllocator”.

In current architecture, we can also add more FPGA vendor plugins, ie, for AWS FPGA instance. The “AbstractFpgaVendorPlugin” interfaces provide abstractions that keep minimum changes to existing code.

2.1 NM Java side

The process of how FpgaResourcePlugin gets in is as follows:

1. FpgaResourcePlugin is created based on “yarn.nodemanager.resource-plugins=yarn.io/fpga”
2. When initializing, FpgaResourcePlugin created one vendor plugin based on “yarn.nodemanager.resource-plugins.fpga.vendor-plugin.class” or use “IntelFpgaOpenclPlugin” by default.
3. FpgaResourcePlugin creates FpgaResourceHandler when LCE creating
4. FpgaResourcePlugin sets vendor plugin to FpgaDiscoverer, and in initializing of FpgaDiscoverer, the vendor plugin gets initialized
5. FpgaResourcePlugin will initialize FpgaNodeResourceUpdateHandler which will use FpgaDiscoverer to update the FPGA resources reported to RM

One thing to note is that, only the vendor plugin initialization exception will fail the NM. We won’t expect vendor plugin’s other interfaces break the NM since it’s platform dependent and can hardly be fully trusted.

So far, the key components gets initialized. And when LCE wants to launch container, the FpgaResourceHandler will do preStart callback. This should be familiar to us and won’t be explained here.

2.2 Native side FPGA module

Currently, we’ve not found any new requirements besides native cgroups device controler changes, so the interfaces is almost same with GPU native module. But we still implement a FPGA native module in case we’ve new features in the future.

```
container-executor fpga --excluded_fpgas=0,1 --container_id=container_x_y_z
```

3. Test environment and result

We’ve done the unit test to cover the component logic. And for the end to end tests, we choose a modified distributed shell and apache spark to run the sample FPGA application. The hardware and software test environment are as follows:

Hardware environment	A desktop with: Intel(R) Core(TM) i7-6700 Nallatech 385a
Software environment	RedHat 6.8 Intel FPGA SDK for OpenCL Nallatech BSP Hadoop 3.1.0-SNAPSHOT Modified Distributed-Shell requesting FPGA Modified Spark requesting FPGA