Fallacies and Pitfalls in HPC and Big Data Convergence

Wednesday, November 6th, 2019

Plan

Why HPC and Big Data Should Meet

Two main reasons:

- Money: two distributed systems infrastructure: twice the cost to
  - Deploy
  - Enhance
  - Maintain
  - Operate
- Science:
  - Life science, genomics, astronomy, HEP ... have huge datasets and may perform some HPC on it. Some of their applications may become HPC/Big Data standards
  - Possibilities may let new applications appear

Why HPC and Big Data Did Not Really Meet yet

Some projects so far have addressed the problem - mainly based on Hadoop + MPI with no real large adoption:

- Integrating HPC tools into Hadoop environment (Hamster, ...): most of the time not accurate due to application latency, disappointing results
- Deploying Hadoop on Demand on HPC resources (HoD, ...): most of the time not accurate because of hardware requirements

Many attempts to make Big Data and HPC meet failed because of pitfalls and fallacies hidden at any level:

- Applications
- Middleware
- Hardware

Plan

Fallacy: There is a large, real need for convergence, many applications will benefit about this

HPC applications

- Weather forecast,
- Molecular dynamics,
- Numerical simulation
- ...

Big Data applications

- Market analysis
- Recommender systems
- IoT monitoring,
- ...

- Big Data and HPC core markets are different!
- Most of Big Data applications are fast-evolving because of the evolving nature of data available, while HPC applications are more stable
- Real HPC/Big Data applications exists, but they won't represent the core of HPC and Big Data development. Maybe some will appear later.

Pitfall: Programmer team for convergence

HPC programmer

- numerical analysts,
- Codes in C/C++,
- Thinks imperative
- ...

Big Data programmer

- data scientist
- code in Python/Java
- Thinks functional
- ...

- Big Data and HPC programmers are both computer scientists, but totally different ones
- two different worlds are complicated to coordinate
- Thus, a more efficient way to work is to isolate both worlds, and reuse "black-box" libraries

Plan

Pitfall: HPC+Big Data hardware

Nowadays HPC and Big Data platforms are different:

- HPC Storage = SAN, Big Data = local, cheap disks
- HPC and Big Data platforms are not of the same scale: billion of cores vs. thousands of machines
- HPC tries to make full use of CPU/GPU, Big Data is focused on I/O.

Fallacy: There will be a perfect hardware to address convergence

Designing a parallel system is often done for a specific kind of application that can be:

- IO-bound
- CPU-bound
- Memory-bound

As HPC is CPU-bound and Big Data is IO-bound, it means that money has to be invested on both sides to be to the top:

- Hybrid architecture will be by design something not too bad at both, but not perfect at both.
- As an example, big data request high throughput on IO and HPC request low latency. It will be difficult to address both at the same time.

Plan

Pitfall: Scheduling theory

Nowadays HPC and Big Data relies on different theoretical ground:

- HPC performs most of the time a mix of task and data parallelism
- Inter-tasks communications are really different.
- Big Data have a tendency to treat all nodes as equals. MapReduce for example has good scaling property when all nodes perform a Map simultaneously.
- Big Data relies on statistics and good probabilistic properties of data distribution, while HPC code explicitly distribute data

Things are fast changing and some Big Data environment (Hadoop for example) now include more support for task parallelism, but there's still an enormous gap between the two.

Pitfall: Middleware

Nowadays HPC and Big Data stacks are different:

- Performances: Big Data exhibits larger latency compared to HPC. BD scaling implies often larger tasks.
- HPC is optimized since almost 50 years, 10 years for BigData
- HPC code is often optimized for its hardware platform, while BigData is not that deep into the architecture
- Languages and philosophy used in both may be different

Many projects have failed due to the complexity of integrating both stacks.

What can be learned from all of this?

- It's a bad idea to use Big Data tools on HPC resources: as the scaling "bottleneck" is Big Data, big data platform may be the underlying architecture.
- HPC tools suffer from the on-demand resource philosophy of Hadoop: HPC tools must be deployed on long-lived resources, not on volatile containers.
- It's complicated to integrate such complex and fast evolving stacks into one: they should avoid dependencies between each other.
- If all other locks can be opened, scheduling will have to be considered carefully.

We then decided to build a prototype working on Hadoop platform, with non-volatile containers, working at the workflow level, reusing library as closed source

Plan

YML

YML ?

- French research project active since more than 20 years
- Worflow engine relying on an XML-based language to describe applications
- Support for multi-architectural level of parallelism (support for Partitioned Global Address Space Languages such as XcalableMP (XMP))
- Used on supercomputers around the world, in Europe, Japan and US.

Features:

- Workflow programming
- parallel and distributed programming
- shared-memory parallel programming/accelerator

YML basics

Two main parts:

- Frontend is platform-agnostic: it provides end users ways to develop applications
- Backend is platform-specific: it provides implementation for tasks and actual scheduling on the platform

Application development and runtime:

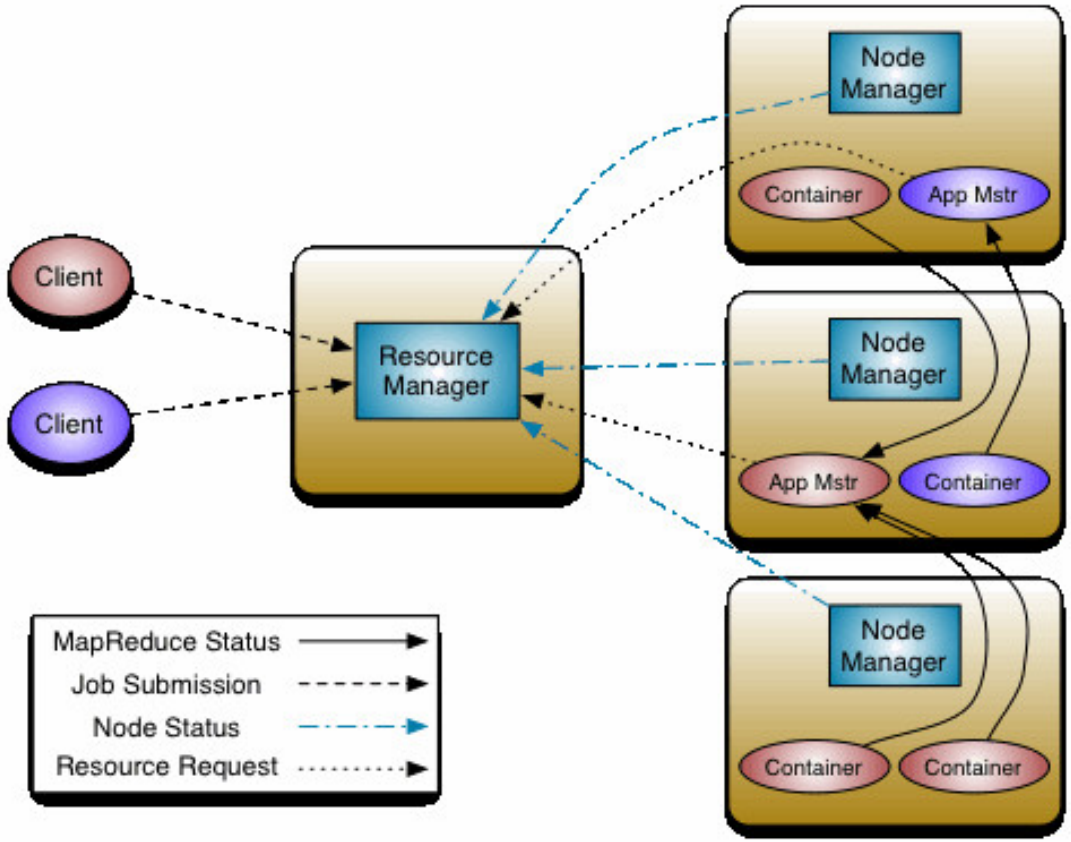
- An YML application is described as a DAG-based program composed of abstract components.
- At runtime, implementation component have to be provided for the abstract component used and the corresponding platform
- Worflow engine then starts to run on abstract component and schedules them by sending which tasks may be ready to be started by a backend scheduler.

Hadoop Overview

Main idea: separating global resource management and application inner management.

- A unique Resource Manager
  - Handles client requests and fair resource allocation to users
  - Allocates (Docker-like) Containers
- For each application, an Application Master (AM) is running:
  - Manages tasks (monitor, scheduling)
  - Asks RM resources and receive it in a Container
  - Running in a Container itself
  - Inside AM Execution Engine manages scheduling and resource allocation
- For each node, a NodeManager handles containers and interacts with RM for monitoring.

YARN Overview

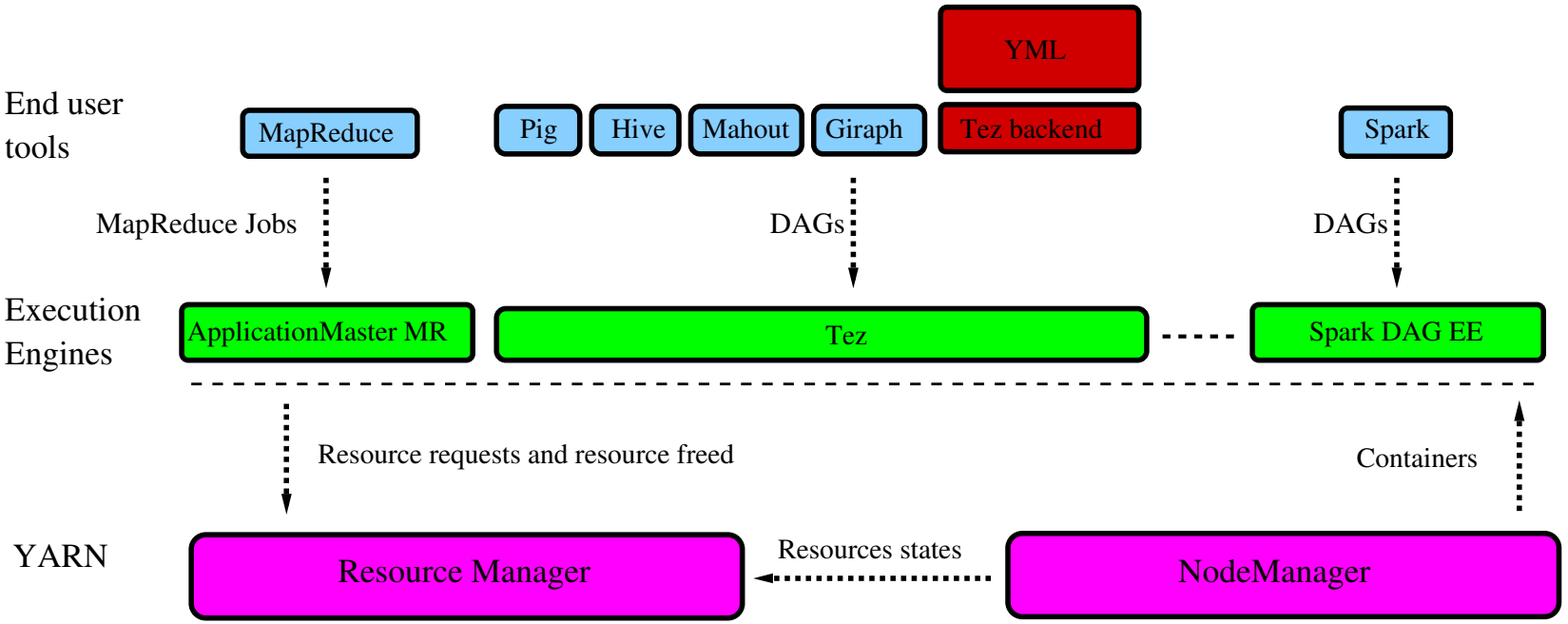


(source: Apache)

Tez Execution Engine

- Manages communications with RM: asks for resources and release them, handle preemption
- Manages the set of Containers already allocated to it
- Handles DAG
  - Eventually release them when useless
  - Asks for new ones if necessary
- Schedules tasks or balance load into Containers
- Makes decisions based on information of headroom given by RM, may not have knowledge about the whole platform.

Hadoop + YML: The big picture



(Work in Progress)

Conclusion

Thanks for your attention!