HPC Scheduling with Kubernetes

Sprint 4

Siyuan Chen, Yilin Xu, Nidhi Shah, Soufiane Jounaid, Juhi Paliwal

Claudia Misale(IBM), Carlos Eduardo Arango Gutierrez (RedHat), Daniel Milroy (Lawrence Livermore National Laboratory)

Content

- Recap of the project
- Introduction to the state synchronization problem
- Potential solution: Operator
- Demo of the Operator
- Accomplished so far
- Burndown chart
- Plans for next sprint

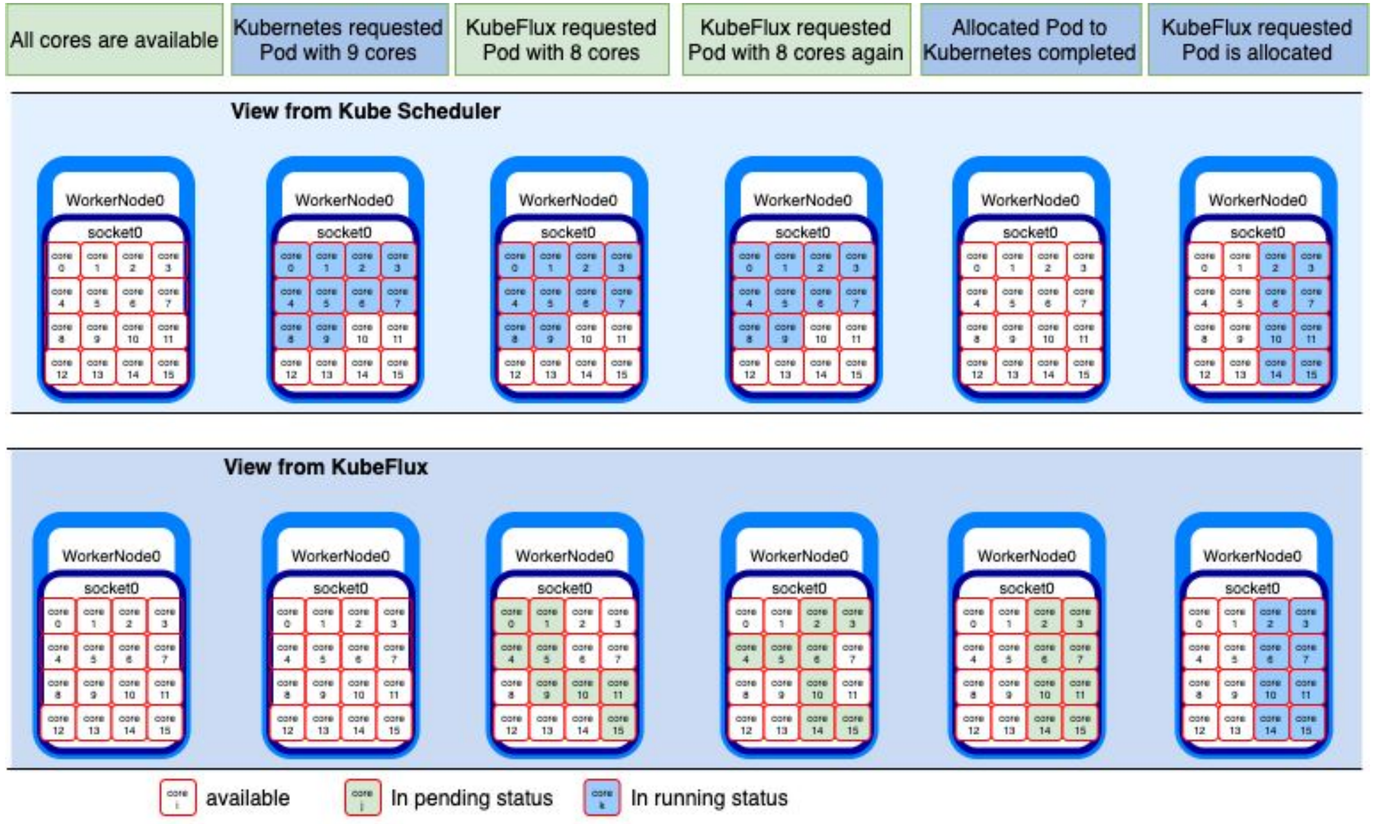
About the Project so far

- State inconsistencies between the Kubernetes cluster and the Kube-Flux scheduler. This hinders utilization of the cluster, queued jobs wait forever.
  - Implemented an informer that carries updated cluster state information to the Kube-Flux resource graph for its own scheduled pods.
  - Exploring different solutions to the state synchronization (co-scheduling) issue for pods scheduled by multiple schedulers.

Informer Solution

- KubeFlux schedules the pods on any available worker nodes. However it is not aware of any change in status of these allocated pods.
- This prohibits KubeFlux to request allocation of new pods on the same worker node.
- The implemented solution uses an informer that frequently watches the pods scheduled by KubeFlux and maintains the correct status of its pods.
- This solves the inconsistency of the state of resources at KubeFlux for its own scheduled pods.
- However, there still exists a problem with the pods scheduled by any other schedulers.

Synchronization Problem across Schedulers


All cores are available
Kubernetes requested Pod with 9 cores
KubeFlux requested Pod with 8 cores
KubeFlux requested Pod with 8 cores again
Allocated Pod to Kubernetes completed
KubeFlux requested Pod is allocated
View from Kube Scheduler
View from KubeFlux
WorkerNode0
socket0
core
available
In pending status
In running status

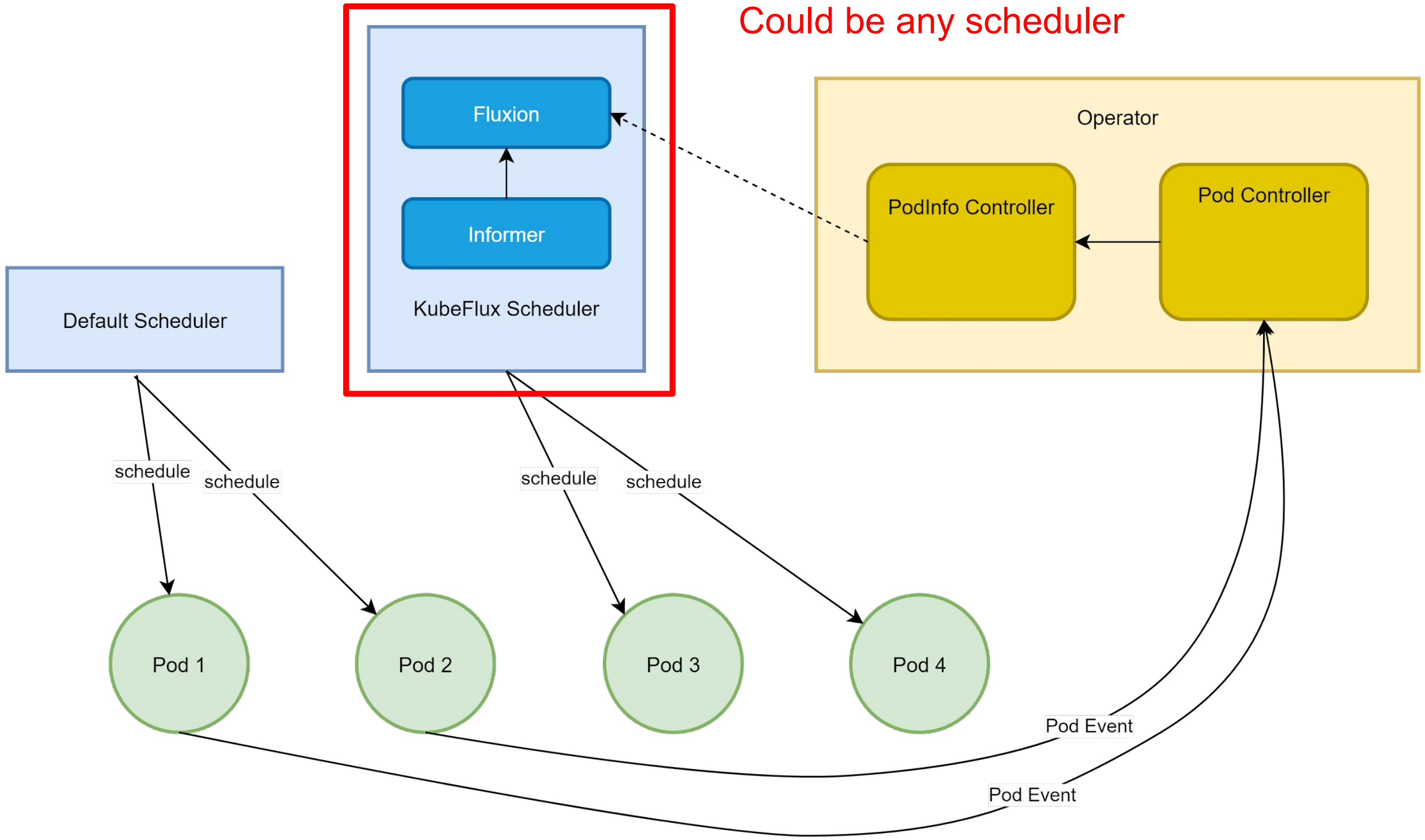
Cluster has 1 worker node with 1 cpu socket. It has 16 cores.

Kublet manages the allocation of pods for any scheduler and is aware of all the occupied resources.

Main conceptual challenges

- Abstraction: Enabling co-scheduling of any two scheduler plugins.
  - The solution should offer a representation of how it records changes
  - Schedulers wanting to make use of co-scheduling to become aware of other schedulers can translate the changes recorded by the operator and update their own internal state accordingly.
  - Each component participating in co-scheduling is specific to a certain level. The interfaces have to be prescriptive of how to implement co-scheduling but not too restrictive.

Potential Solution: Pod Controller



Main technical challenges

- Kubernetes has 3 different ways of specifying pod resource needs.
    - Priority 1: Guaranteed Pod
    - Priority 2: Burstable Pod
    - Priority 3: Best-Effort Pod
- Resources usage fluctuations
- Imperative HPC management philosophy doesn’t align with the declarative K8s management philosophy

Demo Outline

1. Demo description
2. Deploy CRD and Operator
3. Run Pi Test with Default Scheduler
4. Check Generated Custom Resources

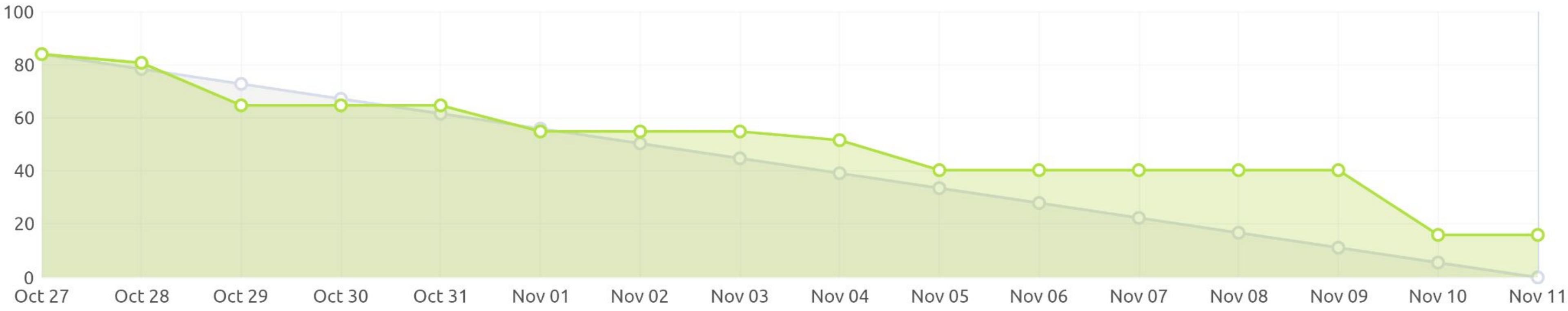
Demo Description

- Environment: Cluster running on local machine
  - 2 nodes
    - 1 Master node: Kubernetes control plane
    - 1 worker node
      - 16 virtual cores in 1 socket and 1 memory node
- Tools:
  - Kind
  - Docker
- Task:
  - Deploy our operator and then deploy Pi calculation program
- Goal
  - Demonstrate Custom Resources are generated for pods running on a worker node

Accomplished So Far

- Define CustomResourceDefinition (CRD)
- Test CRD with YAML file manually
- Create an operator through operator-sdk

Burndown Chart



Needs Info:

- Perform performance tests on a large cluster
- Analyze performance test results

Plans for Next Sprint

- Demo informer in an open shift cluster (real world environment)
- Performance analysis of the Pod Informer