Cloud Native Batch Computing Platform Volcano

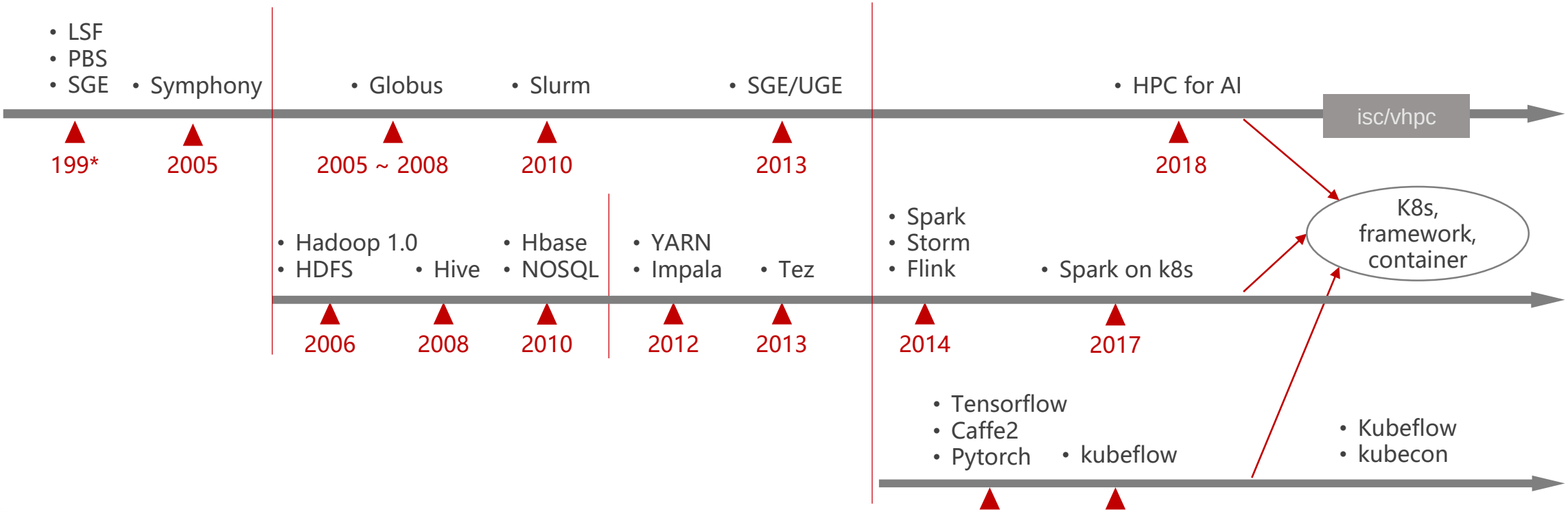
Yang Wang, Contributor of Volcano Community

contents

01

Volcano Project Introduction

Trends of HPC, Big Data, and AI Batch Computing



1. **New framework and ecosystem** is possible since Spark
2. YARN, Mesos, Kubernetes are known as **resource manager**
3. HPC application would like to leverage container (ISC/**VHPC**)
4. New bigdata framework, e.g. Spark, start to support kubernetes
5. Most AI application are built on Kubernetes natively

Batch on K8s: Challenges

Job management

- Pod level scheduling, no awareness of upper-level applications.
- Lack of fine-grained lifecycle management.
- Lack of task dependencies, job dependencies.

mainstream computing framework support

- Insufficient support for mainstream computing frameworks like mpi, tensorflow, mxnet, pytorch.
- Complex deployment and O&M because each framework conrespending to a different operator.

Scheduling

- Lack of job based scheduling, e.g. job ordering, job priority, job preemption, job fair-share, job reservation.
- Not enough advanced scheduling algorithms, E.g. CPU topology, task-topology, IO-Awareness, backfill.

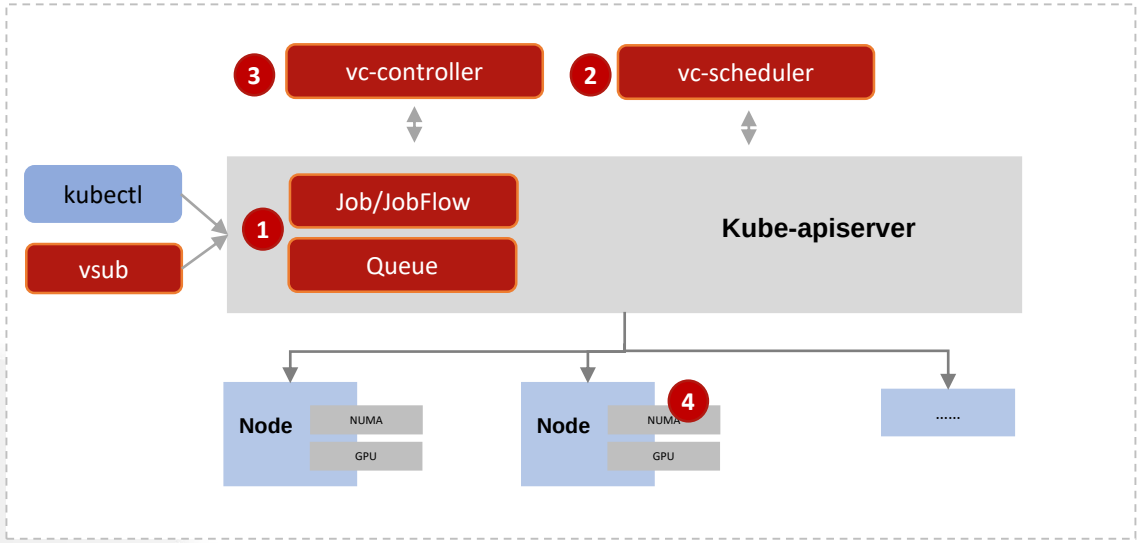
Resource planning, sharing, heterogeneous computing

- Lack of support to resource sharing mechanism between jobs, queues, namespaces.
- Lack of Deeper support on heterogeneous resources.

Performance

- Not enough throughput, roundtrip for batch workload.

Project introduction



Current Status

- The industry's first cloud native batch computing platform
- Open-sourced in June 2019, entered CNCF in 2020, and listed as a CNCF incubation project
- 3,100 stars and 500+ global contributors
- Application to 50+ enterprises

Key Features

- **Unified Job Management**

  Provides complete job lifecycle management and supports almost all mainstream compute frameworks, such as Pytorch, MPI, Horovod, Tensorflow, Spark-operator, Flink-operator.
- **Rich Advanced Scheduling Strategies**

  Supports fair scheduling, task topology scheduling, SLA-based scheduling, job preemption, backfilling, flexible scheduling, and mixed deployment.
- **Fine-Grained Resource Management**

  Provides job queue, queue resource reservation, queue capacity management, and multi-tenant dynamic resource sharing.
- **Performance Optimization and Heterogeneous Resource Management**

  Supports scheduling performance optimization, Kubernetes-based scalability, throughput, network, and runtime optimization, and heterogeneous hardware such as x86, ARM, GPU, Ascend, Kunlun.

VOLCANO CLOUD NATIVE COMPUTING FOUNDATION THE LINUX FOUNDATION 开源软件学园

Volcano Job

Volcano Job:

Unified type interface, supporting mainstream job types in the industry, such as mpi, pytorch, tensorflow, etc.

- ***schedulerName*** specifies the name of the scheduler used
- ***minAvailable*** means that only when the minAvailable number of Pod resources satisfy the job will they be scheduled together
- ***plugins***: Extend the Job API and define customized requirements
- ***policies***: Configure job lifecycle management policies
- ***tasks.name*** Specifies the type of Task in the Job
- ***tasks.replicas*** Specifies the number of copies of the Task

```yaml
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: mpi-job
  labels:
    "volcano.sh/job-type": "MPI"
spec:
  # minimum number of pods need to be started
  minAvailable: 3
  schedulerName: volcano
  plugins:
    # job level ssh trust
    ssh: []
    # define network relevant info for running,
    # hosts, headless services etc.
    svc: []
  # restart who job if any pod get evicted
  policies:
    - event: PodEvicted
      action: RestartJob
  tasks:
    - replicas: 1
      name: mpimaster
      # Mark whole job completed when mpiexec completed
      policies:
        - event: TaskCompleted
          action: CompleteJob
```

Multiple Pod template

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: tensorflow-dist-mnist
  labels:
    "volcano.sh/job-type": “tensorflow"
spec:
  # minimum number of pods need to be started
  minAvailable: 6
  schedulerName: volcano
  tasks:
    - replicas: 2
      name: ps
      template:
        spec:
          containers:
            ……
            image: volcanosh/dist-mnist-tf-example:0.0.1
    - replicas: 4
      name: worker
      template:
        spec:
          containers:
            ……
          image: volcanosh/dist-mnist-tf-example:0.0.1
```

Scenes:

Most batch computing workloads include multiple different task types, such as TensorFlow (ps/worker), MPI (master/worker). At the same time, the images used by different task types require different resources, such as Tensorflow's PS mainly uses CPU resources, while workers tend to use GPU resources for acceleration.

policies: Configure job lifecycle management policies

tasks.name Specifies the type of Task in the Job

Volcano Job Plugin

volcano-job-yaml

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: mpi-job
spec:
  minAvailable: 3
  schedulerName: volcano
  plugins:
    ssh: []
    svc: []
    env: []
    mpi: []
    pytorch: []
    tensorflow: []
```

Volcano job plugin example

volcano-scheduler-configmap

```
actions: "enqueue, allocate, backfill"
tiers:
- plugins:
  - name: priority
  - name: gang
  - name: conformance
- plugins:
  - name: overcommit
  - name: drf
  - name: predicates
  - name: nodeorder
  - name: proportion
  - name: binpack
```

volcano scheduler plugin example

- Only valid for the volcano job that sets this field;
- It can only be used for volcano job type jobs and cannot be applied to other workloads;
- Role: Help volcano support upstream computing frameworks, including MPI, Tensorflow, Pytorch, etc.;

- It will take effect for all jobs scheduled using volcano;
- Configured in the volcano-scheduler-configmap;
- Function: Declare the scheduling strategy in the scheduler;

02

Application of Volcano in AI, HPC, Big Data

01 Pytorch Job

02 TensorFlow Job

03 MPI job

04 Spark job

Pytorch Job

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: pytorch-job
  labels:
    "volcano.sh/job-type": “Pytorch“

spec:
  minAvailable: 1
  schedulerName: volcano

  plugins:
    pytorch: ["--master=master","--worker=worker","--port=23456"]

  policies:
    - event: PodEvicted
      action: RestartJob
```

Set job types according to business needs

Configure the role and port information of the Pytorch job through the general plugin provided by Volcano

Configure Job Restart Policy

Pytorch Job

```
tasks:
  - replicas: 1
    name: master
    policies:
      - event: TaskCompleted
        action: CompleteJob
    template:
      spec:
        containers:
          - image: gcr.io/kubeflow-ci/pytorch-dist-sendrecv-test:1.0
            imagePullPolicy: IfNotPresent
            name: master
        restartPolicy: OnFailure
  - replicas: 2
    name: worker
    template:
      spec:
        containers:
          - image: gcr.io/kubeflow-ci/pytorch-dist-sendrecv-test:1.0
            imagePullPolicy: IfNotPresent
            name: worker
            workingDir: /home
        restartPolicy: OnFailure
```

Configure job termination policies. The entire Pytorch job is considered complete when the pytorch master is complete.

Worker role configuration

Pytorch Demo

Pytorch job status:

```
[root@ecs-4b42-0002 demo-kind-test]# kubectl get pod
NAME                   READY   STATUS    RESTARTS   AGE
pytorch-job-master-0   1/1     Running   0          4s
pytorch-job-worker-0   1/1     Running   0          4s
pytorch-job-worker-1   1/1     Running   0          4s
```

Pytorch job scheduling complete

```
[root@ecs-4b42-0002 demo-kind-test]# kubectl get pod
NAME                   READY   STATUS      RESTARTS   AGE
pytorch-job-master-0   0/1     Completed   0          36s
pytorch-job-worker-0   0/1     Completed   0          36s
pytorch-job-worker-1   0/1     Completed   0          36s
```

Pytorch job run complete

```
[root@ecs-4b42-0002 demo-kind-test]# kubectl logs pytorch-job-master-0
INFO:root:Torch version: 1.0.0
INFO:root:MASTER_PORT: 23456
INFO:root:MASTER_ADDR: pytorch-job-master-0.pytorch-job
INFO:root:WORLD_SIZE: 3
INFO:root:RANK: 0
INFO:root:Result from worker 1 : tensor([[0.4862, 0.4732],
        [0.7807, 1.2848]])
INFO:root:Result from worker 2 : tensor([[0.4862, 0.4732],
        [0.7807, 1.2848]])
```

Job log output

01 Pytorch Job

02 TensorFlow Job

03 MPI Job

04 Spark Job

TensorFlow Job

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: tensorflow-benchmark
  labels:
    "volcano.sh/job-type": "Tensorflow"

spec:
  minAvailable: 3
  schedulerName: volcano
  plugins:
    env: []
    svc: []

  policies:
    - event: PodEvicted
      action: RestartJob
```

Set job types according to business needs

Enable SSH password-free authentication

Create a headless service to solve the communication problem between ps and worker

Configure Job Restart Policy

TensorFlow Job

```
tasks:
    - replicas: 1
      name: ps
      template:
        spec:
          imagePullSecrets:
            - name: default-secret
          containers:
            - command:
                - sh
                - -c
                - |
                  PS_HOST=`cat /etc/volcano/ps.host | sed 's/$/&:2222/g' | tr "\n" ","`;
                  WORKER_HOST=`cat /etc/volcano/worker.host | sed 's/$/&:2222/g' | tr "\n" ","`;
                  python tf_cnn_benchmarks.py --batch_size=32 --model=resnet50 --
variable_update=parameter_server --flush_stdout=true --num_gpus=1 --
local_parameter_device=cpu --device=cpu --data_format=NHWC --job_name=ps --
task_index=${VK_TASK_INDEX} --ps_hosts=${PS_HOST} --worker_hosts=${WORKER_HOST}
              image: volcanosh/example-tf:0.0.1
              name: tensorflow
              ports:
                - containerPort: 2222
                  name: tfjob-port
              workingDir: /opt/tf-benchmarks/scripts/tf_cnn_benchmarks
            restartPolicy: OnFailure
```

PS running configuration

Expose port 2222

TensorFlow Job

```
- replicas: 2
    name: worker
    policies:
      - event: TaskCompleted
        action: CompleteJob
    template:
      spec:
        imagePullSecrets:
          - name: default-secret
        containers:
          - command:
              - sh
              - -c
              - |
                PS_HOST=`cat /etc/volcano/ps.host | sed 's/$/&:2222/g' | tr "\n" ","`;
                WORKER_HOST=`cat /etc/volcano/worker.host | sed 's/$/&:2222/g' | tr "\n" ","`;
                python tf_cnn_benchmarks.py --batch_size=32 --model=resnet50 --
variable_update=parameter_server --flush_stdout=true --num_gpus=1 --
local_parameter_device=cpu --device=cpu --data_format=NHWC --job_name=worker --
task_index=${VK_TASK_INDEX} --ps_hosts=${PS_HOST} --worker_hosts=${WORKER_HOST}
            image: volcanosh/example-tf:0.0.1
            name: tensorflow
            ports:
              - containerPort: 2222
                name: tfjob-port
            workingDir: /opt/tf-benchmarks/scripts/tf_cnn_benchmarks
        restartPolicy: OnFailure
```

Configure job termination policies

worker running configuration

Expose port 2222

TensorFlow Demo

Tensorflow job status:

```
NAME                            READY   STATUS    RESTARTS   AGE
tensorflow-benchmark-ps-0       1/1     Running   0          6s
tensorflow-benchmark-worker-0   1/1     Running   0          6s
tensorflow-benchmark-worker-1   1/1     Running   0          6s
```

Tensorflow job scheduling is complete and starts running

```
2023-07-03 11:25:20.462920: I tensorflow/core/distributed_runtime/rpc/grpc_server_lib.cc:324] Started server with tar
get: grpc://localhost:2222
TensorFlow:  1.5
Model:       resnet50
Mode:        training
SingleSess:  False
Batch size:  32 global
             32 per device
Devices:     ['/job:worker/task:0/cpu:0']
Data format: NHWC
Optimizer:   sgd
Variables:   parameter_server
Sync:        True
==========
Running parameter server 0
```

Job log output

01 Pytorch Job

02 TensorFlow Job

03 MPI Job

04 Spark Job

MPI Job

```
metadata:
  name: mpi-job
  labels:
    "volcano.sh/job-type": "MPI"

spec:
  minAvailable: 3
  schedulerName: volcano
  plugins:
    mpi: ["--master=mpimaster","--worker=mpiworker","--port=22"]

  policies:
    - event: PodEvicted
      action: RestartJob
```

Set job types according to business needs

Configure the role and port information of the MPI job through the general plug-in provided by Volcano

Configure Job Restart Policy

MPI Job

```
tasks:
    - replicas: 1
      name: mpimaster
      policies:
        - event: TaskCompleted
          action: CompleteJob
      template:
        spec:
          containers:
            - command:
                - /bin/sh
                - -c
                - |
                  MPI_HOST=`cat /etc/volcano/mpiworker.host | tr "\n" ","`;
                  mkdir -p /var/run/sshd; /usr/sbin/sshd;
                  mpirun --allow-run-as-root --host ${MPI_HOST} -np 2
mpi_hello_world;
              image: volcanosh/example-mpi:0.0.1
              name: mpimaster
              ports:
                - containerPort: 22
                  name: mpijob-port
```

Configure job termination policies. When mpirun completes, the entire MPI job is considered complete.

Master container start command. Host IP mapping files are stored in /etc/volcano. First start the ssh service, then run mpirun to start the mpi job.

Expose port 22

MPI Job

```
tasks:
    - replicas: 2
      name: mpiworker
      template:
        spec:
          containers:
            - command:
                - /bin/sh
                - -c
                - |
                  mkdir -p /var/run/sshd; /usr/sbin/sshd;
              image: volcanosh/example-mpi:0.0.1
              name: mpimaster
              ports:
                - containerPort: 22
                  name: mpijob-port
```

Worker container start command.
Only start the sshd service

Expose port 22

MPI Demo

MPI Job Status:

```
NAME                    READY   STATUS    RESTARTS   AGE
mpi-job-mpimaster-0     1/1     Running   1          22s
mpi-job-mpiworker-0     1/1     Running   0          22s
mpi-job-mpiworker-1     1/1     Running   0          22s
```

MPI job running.
Master pods may restart mid-run due to network build delays.

```
NAME                    READY   STATUS      RESTARTS   AGE
mpi-job-mpimaster-0     0/1     Completed   1
2m12s
```

MPI job ended.
Keep the Master Pod after the job ends, delete the Worker Pod.

```
Warning: Permanently added 'mpi-job-mpiworker-0.mpi-job,10.0.0.157'
(ECOSA) to the list of known hosts.
Warning: Permanently added 'mpi-job-mpiworker-1.mpi-job,10.0.0.159'
(ECOSA) to the list of known hosts.
```

Warning message

```
Hello world from processor mpi-job-mpiworker-0, rank 0 out of 2 processors
Hello world from processor mpi-job-mpiworker-1, rank 1 out of 2 processors
```

Job log output

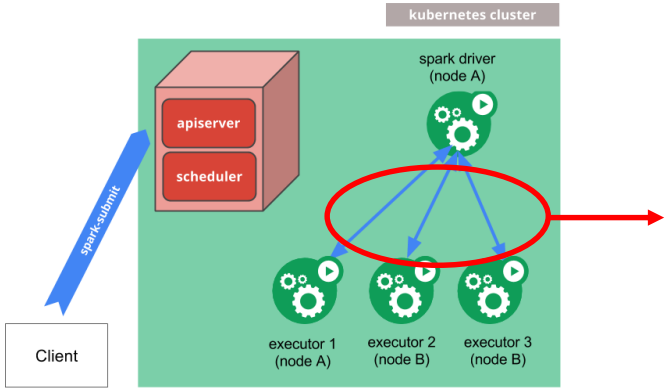
01 Pytorch Job

02 TensorFlow Job

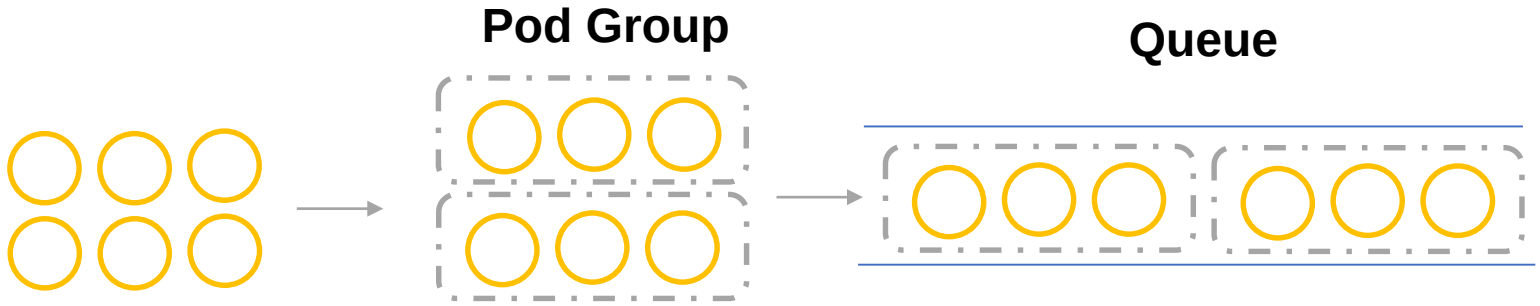
03 MPI Job

04 Spark Job

Scenario: First batch scheduler for Spark on K8s



More k8s native resource support



- Schedule based on **Pod Group**
- Minimal Resource reservation（CPU/MEM）
- Job priority

- Introduce Queue
- Multi-tenants
- Priority
- Fair-share
- Preempt
- ……

1.	✓ Support replicasets/job API	RESOLVED	Holden Karau
2.	Add the ability to specify a scheduler & queue	IN PROGRESS	Apache Spark
3.	Support backing off dynamic allocation increases if resources are "stuck"	OPEN	*Unassigned*
4.	Create a PodGroup with user specified minimum resources required	OPEN	*Unassigned*
5.	✓ Support for specifiying executor/driver node selector	RESOLVED	Yikun Jiang
6.	Support the Volcano Job API	OPEN	*Unassigned*

SPARK-36057: Support volcano/alternative schedulers

- **First Batch scheduler**
  - ✓ Become the first batch scheduler of Spark on Kubernetes in 2022
  - ✓ **1.5K Pod/s** large-scale batch task scheduling capability

VOLCANO CLOUD NATIVE COMPUTING FOUNDATION THE LINUX FOUNDATION 开源软件学园

Spark Job

Podgroup config:

```
apiVersion: scheduling.volcano.sh/v1beta1
kind: PodGroup
spec:
  minMember: 1

  minResources:
    cpu: "4"
    memory: "5Gi“

  priorityClassName: system-node-critical

  queue: default
```

Specify minMember to 1 to make a driver pod

Specify minResources to support resource reservation (the driver pod resource and executors pod resource should be considered)

Spark Job

```
./spark-3.3.1/bin/spark-submit \
--master k8s://https://127.0.0.1:40883 \
--deploy-mode cluster \
--driver-cores 1 \
--driver-memory 2G \
--num-executors 1 \
--executor-cores 1 \
--executor-memory 1G \
--name spark-volcano-wy1 \
--class org.apache.spark.examples.SparkPi \
--conf spark.executor.instances=3 \
--conf spark.kubernetes.container.image=wangyang0616/spark:3.3.1-volcano.v1 \
--conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \
--conf spark.kubernetes.scheduler.name=volcano \
--conf spark.kubernetes.scheduler.volcano.podGroupTemplateFile=…/podgroup-template.yaml \
--conf spark.kubernetes.driver.pod.featureSteps=org.apache.spark.deploy.k8s.features.VolcanoFeatureStep \
--conf spark.kubernetes.executor.pod.featureSteps=org.apache.spark.deploy.k8s.features.VolcanoFeatureStep \
local:///opt/spark/examples/jars/spark-examples_2.12-3.3.1.jar
```

1. Specify custom scheduler
2. Specify scheduler hints (podgroup template)
3. Specify custom feature step

Spark Demo

Spark Job Status:

```
NAME                                          READY   STATUS    RESTARTS   AGE
spark-pi-43882d891f8c6886-exec-1              1/1     Running   0          7s
spark-pi-43882d891f8c6886-exec-2              1/1     Running   0          7s
spark-pi-43882d891f8c6886-exec-3              1/1     Running   0          7s
spark-volcano-wy1-b98f3d891f8c5504-driver     1/1     Running   0          12s
```

The spark driver pod schedules to run.
Driver pod pulls up 3 executor pods.

```
NAME                                          READY   STATUS      RESTARTS   AGE
spark-volcano-wy1-b98f3d891f8c5504-driver     0/1     Completed   0          35s
```

The Spark job runs to completion.

03

Volcano Core Features

01 Job Policy

02 Gang Scheduling

03 Fair Share

04 Priority

05 Preempt

06 Queue

Job Lifecycle Management

Job transactions between phases

From\To	Pending	Aborted	Running	Completed	Terminated
Pending	*	*	*		
Aborted	*	*			
Running		*	*	*	*
Completed				*	
Terminated					*

Error handing

Pod Event list	Action list
PodFailed	AbortJob
PodEvicted	RestartJob
Unknown	TerminateJob
OutOfSync	CompleteJob
CommandIssued	ResumeJob
TaskCompleted	SyncJob

Job status: Job contains multiple different states, and the table shows the transition of different states of the job

After the job is created in the system, the Pod in the job will have different events, such as the Task failure of the MPI Job. The Job life cycle policy is used for user configuration and processing of these events.

Job Policy

```yaml
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: test
spec:
  schedulerName: volcano
  minAvailable: 3
  policies:
    - event: PodEvicted
      action: AbortJob
```

```yaml
tasks:
    - replicas: 1
      name: "ps"
      template:
        spec:
          containers:
            - image: alpine
              command: ["/bin/sh", "-c", "sleep 1000"]
              imagePullPolicy: IfNotPresent
              name: ps
              resources:
                requests:
                  cpu: "1"
          restartPolicy: OnFailure
    - replicas: 2
      name: "worker"
      template:
        spec:
          containers:
            - image: alpine
              command: ["/bin/sh", "-c", "sleep 1000"]
              imagePullPolicy: IfNotPresent
              name: worker
              resources:
                requests:
                  cpu: "1"
          restartPolicy: OnFailure
```

Scenes:

When any pod in the job is evicted, the job fails.

Job Policy Demo

```
NAME           READY   STATUS        RESTARTS   AGE
test-ps-0      1/1     Terminating   0          91s
test-worker-0  1/1     Terminating   0          91s
test-worker-1  1/1     Terminating   0          91s
```

```
[root@ecs-4b42-0002 test-case]# kubectl get vcjob
NAME   STATUS    MINAVAILABLE   RUNNINGS   AGE
test   Aborted   3                         2m20s
```

Evict any pod in the job, the job becomes Aborted, and clean up all pods in the job

Job & Task Policy

```yaml
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: test
spec:
  schedulerName: volcano
  minAvailable: 3
  policies:
    - event: PodEvicted
      action: RestartJob
```

```yaml
tasks:
  - replicas: 1
    name: "ps"
    template:
      spec:
        containers:
          - image: alpine
            command: ["/bin/sh", "-c", "sleep 1000"]
            imagePullPolicy: IfNotPresent
            name: ps
            resources:
              requests:
                cpu: "1"
        restartPolicy: OnFailure
  - replicas: 2
    name: "worker"
    policies:
      - event: PodEvicted
        action: RestartTask
    template:
      spec:
        containers:
          - image: alpine
            command: ["/bin/sh", "-c", "sleep 1000"]
            imagePullPolicy: IfNotPresent
            name: worker
            resources:
              requests:
                cpu: "1"
        restartPolicy: OnFailure
```

Scenes:

In the same job, different types of tasks require different policies, and Volcano allows you to configure policies for specific types of tasks.

Job & Task Policy Demo

```
NAME            READY   STATUS        RESTARTS   AGE
test-ps-0       1/1     Running       0          73s
test-worker-0   1/1     Running       0          73s
test-worker-1   1/1     Terminating   0          73s
```

```
NAME            READY   STATUS    RESTARTS   AGE
test-ps-0       1/1     Running   0          104s
test-worker-0   1/1     Running   0          104s
test-worker-1   1/1     Running   0          7s
```

After the pod of the Worker role is evicted, only the current task is restarted.

```
NAME            READY   STATUS        RESTARTS   AGE
test-ps-0       1/1     Terminating   0          2m23s
test-worker-0   1/1     Terminating   0          2m23s
test-worker-1   1/1     Terminating   0          46s
```

```
NAME            READY   STATUS    RESTARTS   AGE
test-ps-0       1/1     Running   0          7s
test-worker-0   1/1     Running   0          7s
test-worker-1   1/1     Running   0          7s
```

After the pod of the PS role is evicted, restart the entire job.

01 Job Policy

02 Gang Scheduling

03 Fair Share

04 Priority

05 Preempt

06 Queue

Gang

Volcano Job example:

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: gang
spec:
  schedulerName: volcano
  minAvailable: 20
  tasks:
    - replicas: 20
      name: "test"
      template:
        spec:
          containers:
            - image: alpine
              command: ["/bin/sh", "-c", "sleep 1000"]
              imagePullPolicy: IfNotPresent
              name: running
              resources:
                requests:
                  cpu: “1"
          restartPolicy: OnFailure
```

Gang

The Gang scheduling strategy is one of the core scheduling algorithms of the volcano-scheduler. It meets the "All or nothing" scheduling requirements in the scheduling process and avoids the waste of cluster resources caused by arbitrary scheduling of Pods.

Scenes

The Gang scheduling algorithm based on the container group concept is very suitable for scenarios that require multi-process cooperation.

AI scenarios often contain complex processes, such as Data Ingestion, Data Analysts, Data Splitting, Trainer, Serving, Logging, etc., which require a group of containers to work together, which is very suitable for the gang scheduling strategy based on container groups.

The multi-thread parallel computing communication scenario under the MPI computing framework is also very suitable for using the Gang scheduling strategy because the master-slave process needs to work together.

The containers under the container group are highly correlated and there may be resource contention. The overall scheduling and allocation can effectively solve the deadlock.

Gang

Gang scheduling results:

```
NAME              READY   STATUS    RESTARTS   AGE
gang-a-test-0     0/1     Pending   0          16s
gang-a-test-1     0/1     Pending   0          16s
gang-a-test-10    0/1     Pending   0          16s
gang-a-test-11    0/1     Pending   0          16s
gang-a-test-12    0/1     Pending   0          16s
gang-a-test-13    0/1     Pending   0          16s
gang-a-test-14    0/1     Pending   0          16s
gang-a-test-15    0/1     Pending   0          16s
gang-a-test-16    0/1     Pending   0          16s
gang-a-test-17    0/1     Pending   0          16s
gang-a-test-18    0/1     Pending   0          16s
gang-a-test-19    0/1     Pending   0          16s
gang-a-test-2     0/1     Pending   0          16s
gang-a-test-3     0/1     Pending   0          16s
gang-a-test-4     0/1     Pending   0          16s
gang-a-test-5     0/1     Pending   0          16s
gang-a-test-6     0/1     Pending   0          16s
gang-a-test-7     0/1     Pending   0          16s
gang-a-test-8     0/1     Pending   0          16s
gang-a-test-9     0/1     Pending   0          16s
```

The idle resource of the cluster is 12C, the resource request of the current job is 20C in total, all pods in the job are all pending, and no resources are allocated.

01 Job Policy

02 Gang scheduling

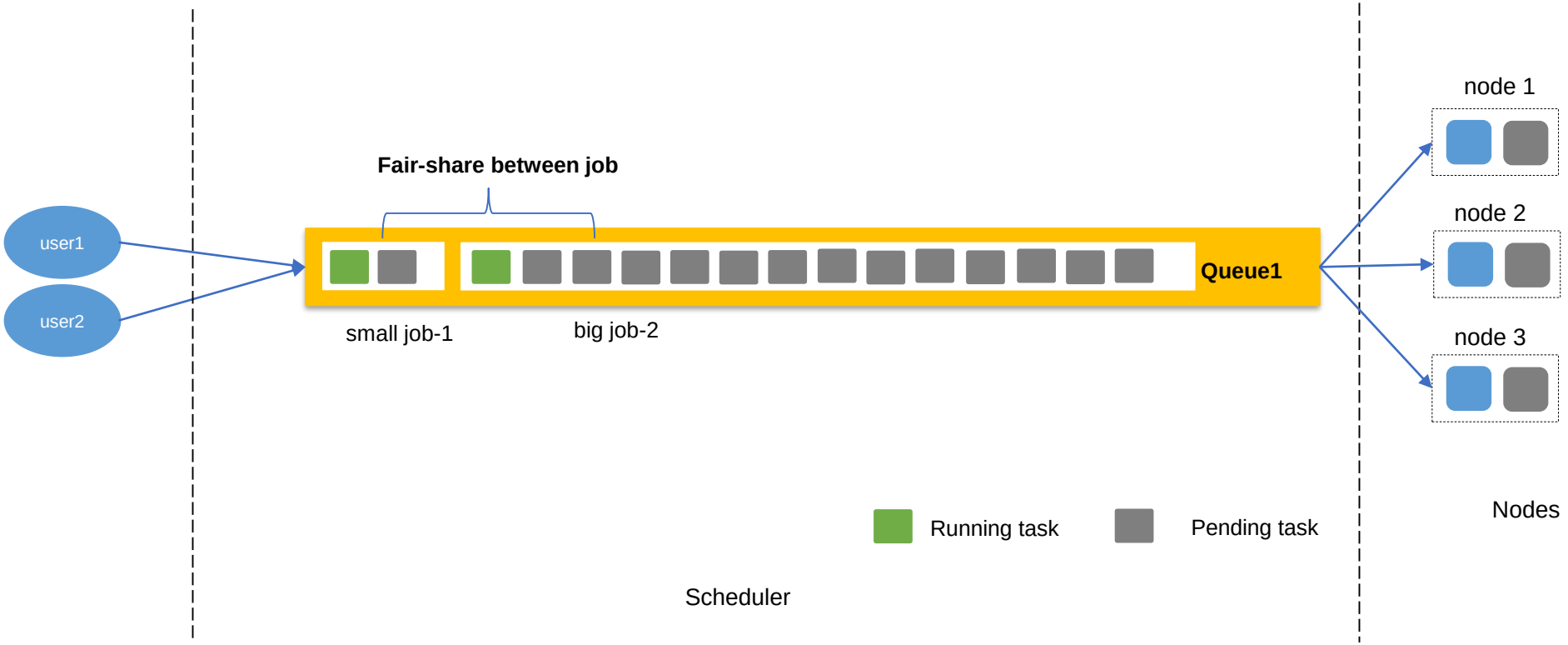
03 Fair Share

04 Priority

05 Preempt

06 Queue

Fair Share



- Resource sharing between jobs
- Queue-level Policy (FIFO, Priority, Fair share, …)

Fair Share

Volcano Job a:

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: job-a
spec:
  priorityClassName: share-1-priority
  schedulerName: volcano
  minAvailable: 2
  tasks:
  - replicas: 12
    name: "test"
    template:
      spec:
        containers:
        - image: alpine
          command: ["/bin/sh", "-c", "sleep 1000"]
          imagePullPolicy: IfNotPresent
          name: running
          resources:
            requests:
              cpu: "1"
        restartPolicy: OnFailure
```

Volcano Job b:

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: job-b
spec:
  priorityClassName: share-2-priority
  schedulerName: volcano
  minAvailable: 2
  tasks:
  - replicas: 6
    name: "test"
    template:
      spec:
        containers:
        - image: alpine
          command: ["/bin/sh", "-c", "sleep 1000"]
          imagePullPolicy: IfNotPresent
          name: running
          resources:
            requests:
              cpu: "1"
        restartPolicy: OnFailure
```

Volcano Job c:

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: job-c
spec:
  priorityClassName: share-3-priority
  schedulerName: volcano
  minAvailable: 2
  tasks:
  - replicas: 48
    name: "test"
    template:
      spec:
        containers:
        - image: alpine
          command: ["/bin/sh", "-c", "sleep 1000"]
          imagePullPolicy: IfNotPresent
          name: running
          resources:
            requests:
              cpu: "1"
        restartPolicy: OnFailure
```

Fair Share Demo

```
[root@ecs-4b42-0002 test-case]# kubectl get pod |grep Pending
job-c-test-10   0/1     Pending   0          2m16s
job-c-test-11   0/1     Pending   0          2m16s
job-c-test-12   0/1     Pending   0          2m16s
job-c-test-13   0/1     Pending   0          2m16s
job-c-test-14   0/1     Pending   0          2m16s
job-c-test-15   0/1     Pending   0          2m16s
job-c-test-16   0/1     Pending   0          2m16s
job-c-test-17   0/1     Pending   0          2m16s
job-c-test-18   0/1     Pending   0          2m16s
job-c-test-19   0/1     Pending   0          2m16s
job-c-test-20   0/1     Pending   0          2m16s
job-c-test-21   0/1     Pending   0          2m16s
job-c-test-22   0/1     Pending   0          2m16s
job-c-test-23   0/1     Pending   0          2m16s
job-c-test-24   0/1     Pending   0          2m16s
job-c-test-25   0/1     Pending   0          2m16s
job-c-test-26   0/1     Pending   0          2m16s
job-c-test-27   0/1     Pending   0          2m16s
job-c-test-28   0/1     Pending   0          2m16s
job-c-test-29   0/1     Pending   0          2m16s
job-c-test-30   0/1     Pending   0          2m16s
job-c-test-31   0/1     Pending   0          2m16s
job-c-test-32   0/1     Pending   0          2m16s
job-c-test-33   0/1     Pending   0          2m16s
job-c-test-34   0/1     Pending   0          2m16s
job-c-test-35   0/1     Pending   0          2m16s
job-c-test-36   0/1     Pending   0          2m16s
job-c-test-37   0/1     Pending   0          2m16s
job-c-test-38   0/1     Pending   0          2m16s
job-c-test-39   0/1     Pending   0          2m16s
job-c-test-40   0/1     Pending   0          2m16s
job-c-test-41   0/1     Pending   0          2m16s
job-c-test-42   0/1     Pending   0          2m16s
job-c-test-43   0/1     Pending   0          2m16s
job-c-test-44   0/1     Pending   0          2m16s
job-c-test-45   0/1     Pending   0          2m16s
job-c-test-46   0/1     Pending   0          2m16s
job-c-test-47   0/1     Pending   0          2m16s
job-c-test-6    0/1     Pending   0          2m16s
job-c-test-7    0/1     Pending   0          2m16s
job-c-test-8    0/1     Pending   0          2m16s
job-c-test-9    0/1     Pending   0          2m16s
```

```
[root@ecs-4b42-0002 test-case]# kubectl get pod |grep Running
job-b-test-0    1/1     Running   0          104s
job-b-test-1    1/1     Running   0          104s
job-b-test-2    1/1     Running   0          104s
job-b-test-3    1/1     Running   0          104s
job-b-test-4    1/1     Running   0          104s
job-b-test-5    1/1     Running   0          104s
job-c-test-0    1/1     Running   0          100s
job-c-test-1    1/1     Running   0          100s
job-c-test-2    1/1     Running   0          100s
job-c-test-3    1/1     Running   0          100s
job-c-test-4    1/1     Running   0          100s
job-c-test-5    1/1     Running   0          100s
```

Job-b and job-c allocate cluster resources fairly, each running 6 copies.

The pending pods are all copies of job-c, and will not allocate more resources because of the many tasks submitted by job-c.

01 Job Policy

02 Gang Scheduling

03 Fair Share

04 Priority

05 Preempt

06 Queue

Priority

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: tensorflow-dist-mnist
  labels:
    "volcano.sh/job-type": “tensorflow"
spec:
  # minimum number of pods need to be started
  minAvailable: 3
  schedulerName: volcano
  priorityClassName: high
  tasks:
    - replicas: 2
      name: ps
      template:
        spec:
          containers:
            ……
            image: volcanosh/dist-mnist-tf-example:0.0.1
    - replicas: 4
      name: worker
      template:
        spec:
          containers:
            ……
          image: volcanosh/dist-mnist-tf-example:0.0.1
```

Scenes:

In the production environment, there are usually some urgent or important application loads, and the SLA of these loads needs to be guaranteed first

Job priority:

The priority represents the importance of the job. High-priority jobs are given priority to obtain cluster resources. When resources are insufficient, high-priority jobs are allowed to preempt low-priority jobs to obtain resources.

Priority

Define high, medium and low priorities:

High priority:

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high-priority
value: 100
globalDefault: false
description: "This priority class should be used for volcano job only."
```

Med priority:

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: med-priority
value: 50
globalDefault: false
description: "This priority class should be used for volcano job only."
```

Low priority:

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: low-priority
value: 10
globalDefault: false
description: "This priority class should be used for volcano job only."
```

Priority

Volcano Job a:

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: priority-high
spec:
  schedulerName: volcano
  minAvailable: 2
  priorityClassName: high-priority
  tasks:
    - replicas: 12
      name: "test"
      template:
        spec:
          containers:
            - image: alpine
              command: ["/bin/sh", "-c", "sleep 1000"]
              imagePullPolicy: IfNotPresent
              name: running
              resources:
                requests:
                  cpu: "1"
          restartPolicy: OnFailure
```

Volcano Job b:

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: priority-medium
spec:
  schedulerName: volcano
  minAvailable: 2
  priorityClassName: med-priority
  tasks:
    - replicas: 12
      name: "test"
      template:
        spec:
          containers:
            - image: alpine
              command: ["/bin/sh", "-c", "sleep 1000"]
              imagePullPolicy: IfNotPresent
              name: running
              resources:
                requests:
                  cpu: "1"
          restartPolicy: OnFailure
```

Volcano Job c:

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: priority-low
spec:
  schedulerName: volcano
  minAvailable: 2
  priorityClassName: low-priority
  tasks:
    - replicas: 12
      name: "test"
      template:
        spec:
          containers:
            - image: alpine
              command: ["/bin/sh", "-c", "sleep 1000"]
              imagePullPolicy: IfNotPresent
              name: running
              resources:
                requests:
                  cpu: "1"
          restartPolicy: OnFailure
```

Priority Demo

```
[root@ecs-4b42-0002 demo-kind-test]# kubectl get pod
NAME                        READY   STATUS    RESTARTS   AGE
priority-low-test-0         0/1     Pending   0          55s
priority-low-test-1         0/1     Pending   0          55s
priority-low-test-10        0/1     Pending   0          55s
priority-low-test-11        0/1     Pending   0          55s
priority-low-test-2         0/1     Pending   0          55s
priority-low-test-3         0/1     Pending   0          55s
priority-low-test-4         0/1     Pending   0          55s
priority-low-test-5         0/1     Pending   0          55s
priority-low-test-6         0/1     Pending   0          55s
priority-low-test-7         0/1     Pending   0          55s
priority-low-test-8         0/1     Pending   0          55s
priority-low-test-9         0/1     Pending   0          55s
priority-medium-test-0      1/1     Running   0          61s
priority-medium-test-1      1/1     Running   0          61s
priority-medium-test-10     1/1     Running   0          61s
priority-medium-test-11     1/1     Running   0          61s
priority-medium-test-2      1/1     Running   0          61s
priority-medium-test-3      1/1     Running   0          61s
priority-medium-test-4      1/1     Running   0          61s
priority-medium-test-5      1/1     Running   0          61s
priority-medium-test-6      1/1     Running   0          61s
priority-medium-test-7      1/1     Running   0          61s
priority-medium-test-8      1/1     Running   0          61s
priority-medium-test-9      1/1     Running   0          61s
```

After deleting high-priority jobs, all idle resources are allocated to medium-priority jobs, and low-priority jobs will not be allocated resources.

01 Job Policy

02 Gang Scheduling

03 Fair Share

04 Priority

05 Preempt

06 Queue

preempt

Priority low job:

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: job-a
spec:
  schedulerName: volcano
  minAvailable: 1
  priorityClassName: low-priority
  tasks:
    - replicas: 12
      name: "test"
      template:
        spec:
          containers:
            - image: alpine
              command: ["/bin/sh", "-c", "sleep 6000"]
              imagePullPolicy: IfNotPresent
              name: running
              resources:
                requests:
                  cpu: "1"
          restartPolicy: OnFailure
```

Priority high job:

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: job-b
spec:
  schedulerName: volcano
  minAvailable: 8
  priorityClassName: high-priority
  tasks:
    - replicas: 8
      name: "test"
      template:
        spec:
          containers:
            - image: alpine
              command: ["/bin/sh", "-c", "sleep 1000"]
              imagePullPolicy: IfNotPresent
              name: running
              resources:
                requests:
                  cpu: "1"
          restartPolicy: OnFailure
```

Two jobs with priority high and low respectively

Number of replicas

Preempt Demo

```
[root@ecs-4b42-0002 test-case]# kubectl get pod
NAME            READY   STATUS    RESTARTS   AGE
job-a-test-0    0/1     Pending   0          8s
job-a-test-1    0/1     Pending   0          8s
job-a-test-10   1/1     Running   0          57s
job-a-test-11   0/1     Pending   0          8s
job-a-test-2    0/1     Pending   0          8s
job-a-test-3    0/1     Pending   0          8s
job-a-test-4    1/1     Running   0          57s
job-a-test-5    1/1     Running   0          57s
job-a-test-6    0/1     Pending   0          8s
job-a-test-7    0/1     Pending   0          8s
job-a-test-8    0/1     Pending   0          8s
job-a-test-9    1/1     Running   0          57s
job-b-test-0    1/1     Running   0          40s
job-b-test-1    1/1     Running   0          40s
job-b-test-2    1/1     Running   0          40s
job-b-test-3    1/1     Running   0          40s
job-b-test-4    1/1     Running   0          40s
job-b-test-5    1/1     Running   0          40s
job-b-test-6    1/1     Running   0          40s
job-b-test-7    1/1     Running   0          40s
```

The high-priority job preempts the resources of the low-priority job, and all 8 copies are running; the low-priority job is expelled from 8 copies, and the remaining 4 copies are running.

01 Job Policy

02 Gang Scheduling

03 Fair Share

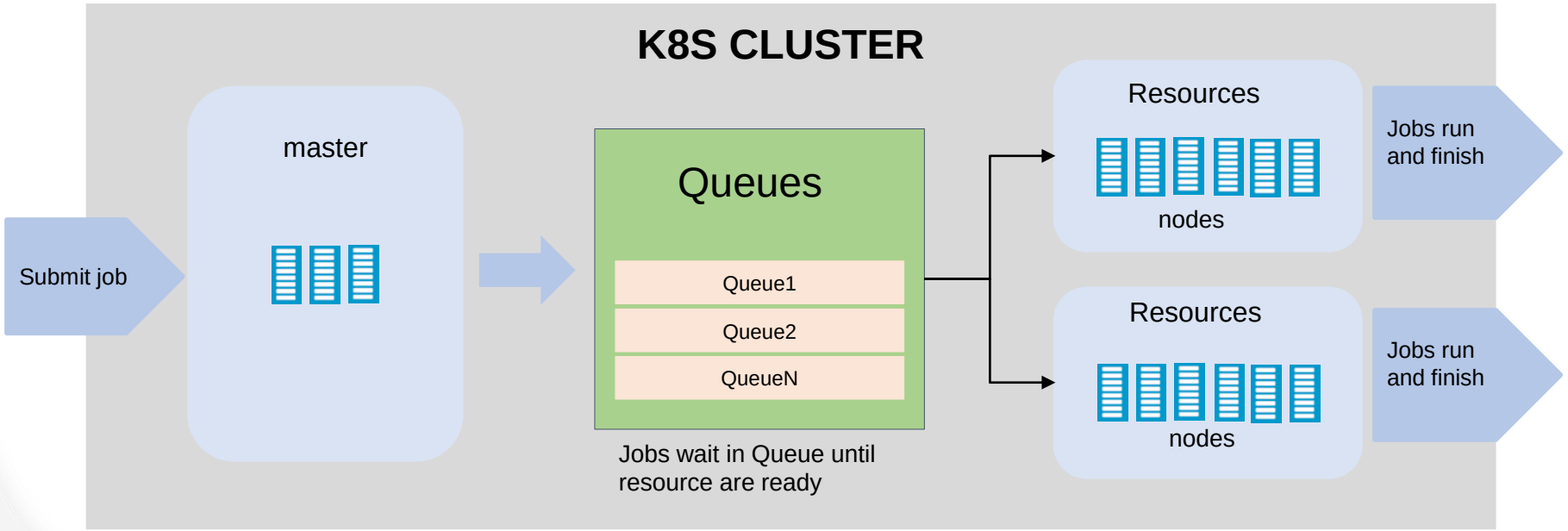
04 Priority

05 Preempt

06 Queue

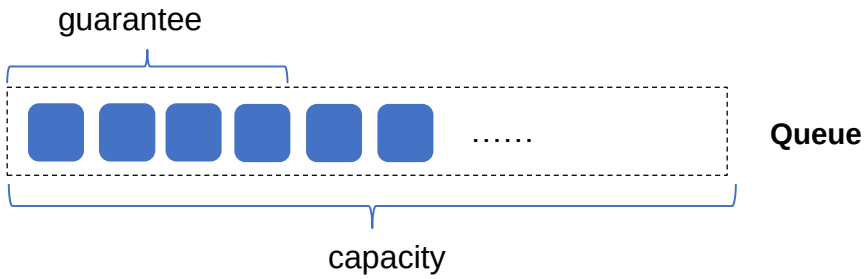
Queue

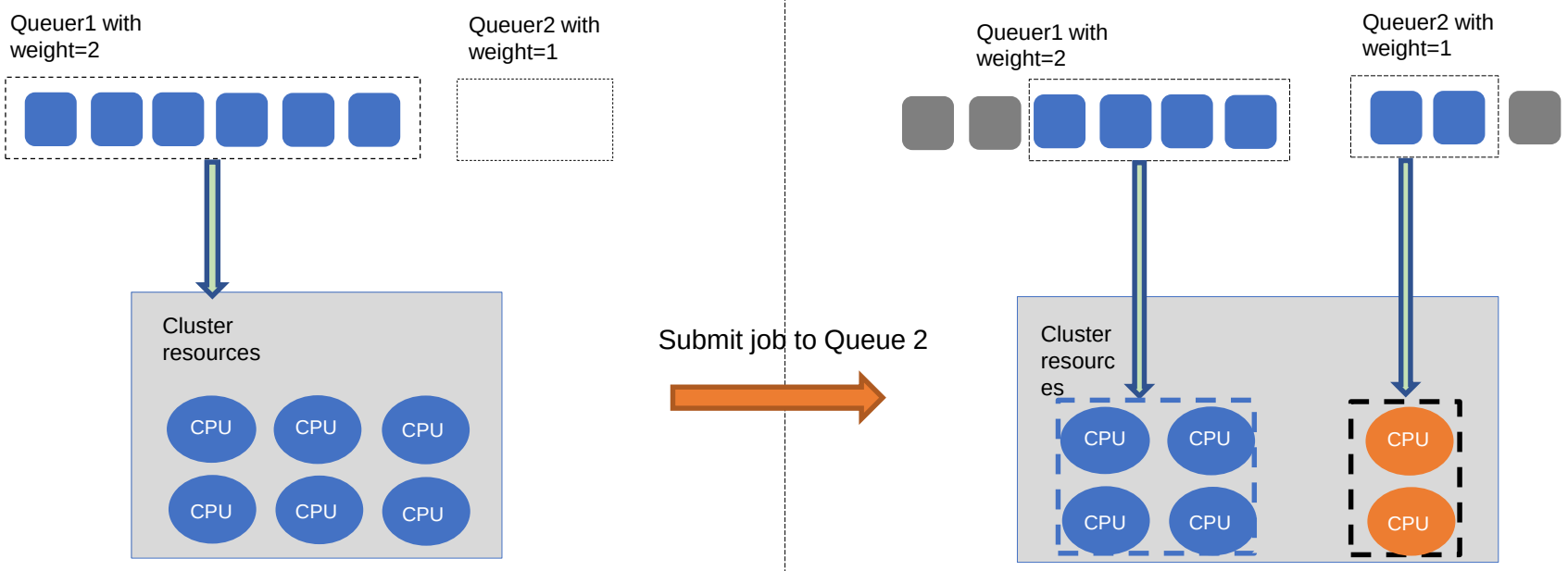
- Cluster-level resource objects, decoupled from user/namespace
- Can be used to share resources between tenants/resource pools



Dynamic Resource Sharing

- Queue resource reservation/queue capacity.
- Provides resource sharing between queues based on weight.





Queue2 is empty. Q1 can borrow resources from Queue2.

Queue2 has workload, it will reclaim resources from Queue1.

Queue Share

Queue a:

```
apiVersion: scheduling.volcano.sh/v1beta1
kind: Queue
metadata:
  name: queue-a
spec:
  reclaimable: true
  weight: 1
```

Queue b:

```
apiVersion: scheduling.volcano.sh/v1beta1
kind: Queue
metadata:
  name: queue-b
spec:
  reclaimable: true
  weight: 3
```

Volano job:

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: job-a
spec:
  queue: queue-a
  schedulerName: volcano
  minAvailable: 1
  tasks:
    - replicas: 10
      name: "test"
      template:
        spec:
          containers:
            - image: alpine
              command: ["/bin/sh", "-c", "sleep 1000"]
              imagePullPolicy: IfNotPresent
              name: running
              resources:
                requests:
                  cpu: "1"
          restartPolicy: OnFailure
```

Cluster 12C idle resources, queue-a logical allocation of 3C resources (1/4), queue-b logical allocation of 9C resources (3/4).

queue-a submit job application 10C resource.

Queue Share Demo

```
[root@ecs-4b42-0002 test-case]# kubectl get pod
NAME           READY   STATUS    RESTARTS   AGE
job-a-test-0   1/1     Running   0          4s
job-a-test-1   1/1     Running   0          4s
job-a-test-2   1/1     Running   0          4s
job-a-test-3   1/1     Running   0          4s
job-a-test-4   1/1     Running   0          4s
job-a-test-5   1/1     Running   0          4s
job-a-test-6   1/1     Running   0          4s
job-a-test-7   1/1     Running   0          4s
job-a-test-8   1/1     Running   0          4s
job-a-test-9   1/1     Running   0          4s
```

```
Spec:
  Reclaimable:  true
  Weight:       1
Status:
  Allocated:
    Cpu:     10
    Memory:  0
  Reservation:
  Running:  1
  State:    Open
```

```
Spec:
  Reclaimable:  true
  Weight:       3
Status:
  Allocated:
    Cpu:     0
    Memory:  0
  Reservation:
  State:  Open
```

The pod of job-a in queue-a is successfully scheduled. (sharing the resources of queue-b)

Queue Capability

Queue a:

```
apiVersion: scheduling.volcano.sh/v1beta1
kind: Queue
metadata:
  name: queue-a
spec:
  reclaimable: true
  weight: 1
  capability:
    cpu: "4"
```

Volcano job a:

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: job-a
spec:
  queue: queue-a
  schedulerName: volcano
  minAvailable: 3
  tasks:
    - replicas: 3
      name: "test"
      template:
        spec:
          containers:
            - image: alpine
              command: ["/bin/sh", "-c", "sleep 1000"]
              imagePullPolicy: IfNotPresent
              name: running
              resources:
                requests:
                  cpu: "1"
          restartPolicy: OnFailure
```

Volcano job:

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: job-b
spec:
  queue: queue-a
  schedulerName: volcano
  minAvailable: 2
  tasks:
    - replicas: 2
      name: "test"
      template:
        spec:
          containers:
            - image: alpine
              command: ["/bin/sh", "-c", "sleep 1000"]
              imagePullPolicy: IfNotPresent
              name: running
              resources:
                requests:
                  cpu: "1"
          restartPolicy: OnFailure
```

Queue Capability Demo

```
[root@ecs-4b42-0002 test-case]# kubectl get pod
NAME           READY   STATUS    RESTARTS   AGE
job-a-test-0   1/1     Running   0          6m19s
job-a-test-1   1/1     Running   0          6m19s
job-a-test-2   1/1     Running   0          6m19s
job-b-test-0   0/1     Pending   0          4m
job-b-test-1   0/1     Pending   0          4m
```

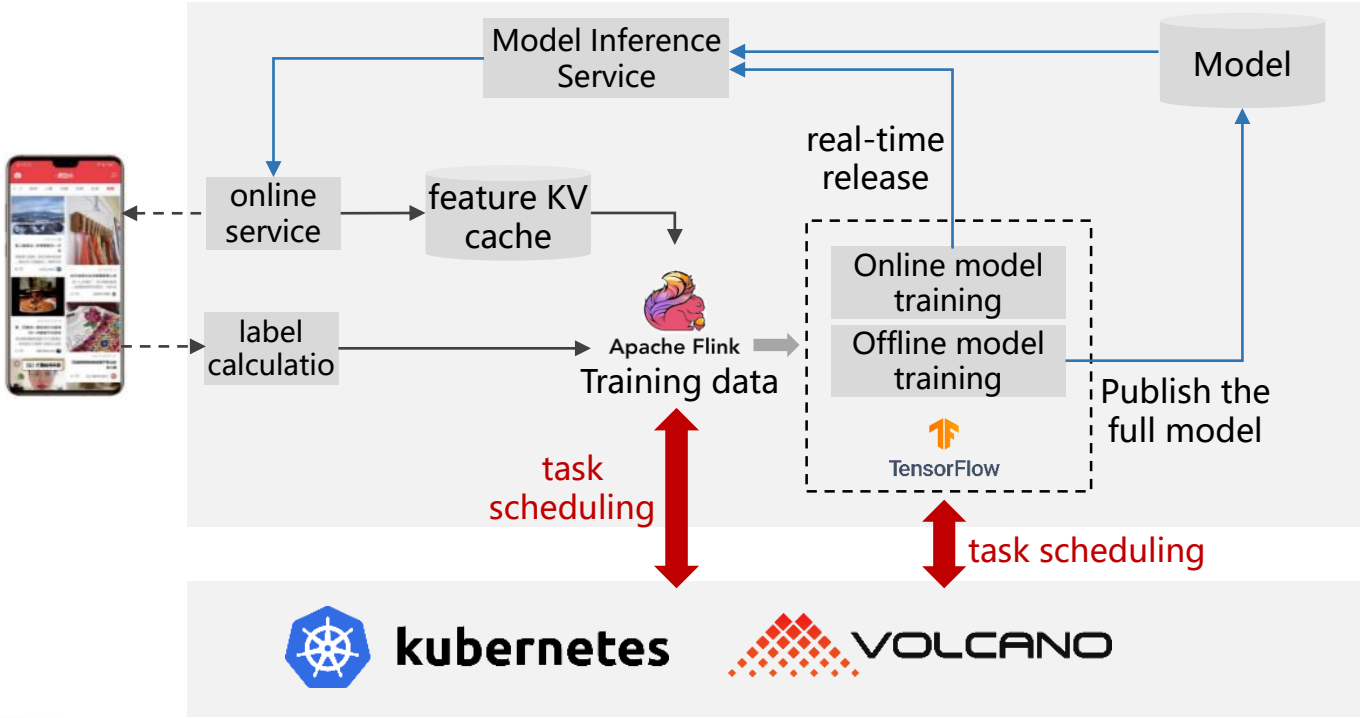
```
Spec:
  Capability:
    Cpu:         4
  Reclaimable:   true
  Weight:        1
Status:
  Allocated:
    Cpu:     3
    Memory:  0
  Inqueue:   1
  Reservation:
  Running:  1
  State:    Open
```

Job-a in queue-a runs successfully, but job-b's application for resources exceeds the hard limit of queue-a's resource upper limit and will not be scheduled.

04

Volcano Use Case

Use Case： AI training performance improved by 20%



*Based on these pain points, we did some research and found Volcano, **which can completely solve our pain points**. Therefore, we also participated in the Kubernetes batch community and became a loyal user of volcano.*

-- Yi Guo(Tech-Lead at Xiaohongshu)

About Xiaohongshu

- *Top social media and e-commerce platform with over **100 million** active users per month.*
- *Recommendation is one of core business. AI platform at Xiaohongshu consists of online and offline training system, which **undertakes hundreds of thousands samples** analysis and model training. The model generation has already be on the **minute scale**.*

Challenge

- Training cluster with **thousands of nodes**
- Recommendation model with nearly **100 billion parameters**
- A single training task contains hundreds of PS and workers
- Require best topology scheduling and performance

Solution

- Adoption of **binpack** and **task-topology** scheduling policy
- Adoption of **gang-scheduling/SLA**

Benefit

- **20%** increase on AI training **speed** overall
- **20%** increase on AI training **throughput**
- Starvation prevention on big jobs

VOLCANO CLOUD NATIVE COMPUTING FOUNDATION THE LINUX FOUNDATION 开源软件学园

Use Case：ING Migrate Big Data Platform from Yarn to K8s

DAP Landing
DAP Genesis

- Platform Entry-point
- Project Management

Buildpacks.io
jupyter
TensorFlow
Spark

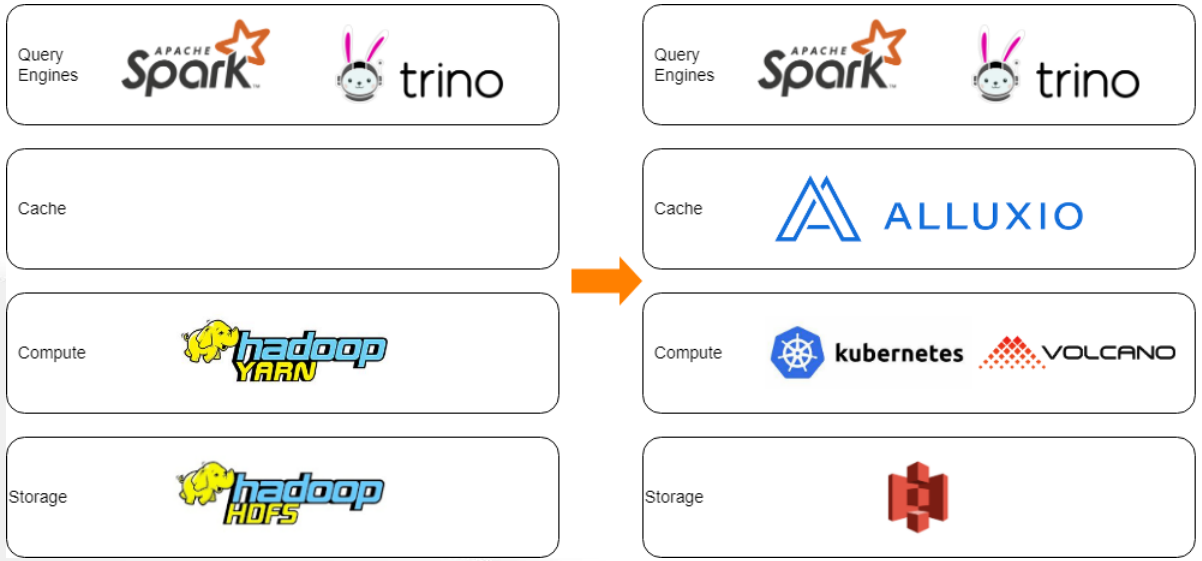
Data Science in a box (Advanced analytics toolbox)

Amundsen

- Data Discovery
- Metadata engine

Apache Superset

- SQL+BI toolset
- Dashboarding



Information reference： https://volcano.sh/en/blog/ing_case-en/

Abort ING:

*ING (International Netherlands Groups) is a world-leading asset management company with services in more than **40 countries**. Its core business is **banking, insurance and asset management**. Introduce cloud-native infrastructure to create a new generation of big data analysis self-service platform.*

Challenge:

- **Unified platform** scheduling for interactive services, resident services, and offline analysis services;
- **Job-level** scheduling management, including life cycle, dependencies, etc.;
- Multi-users allocate resources **fairly and respond quickly** to high-priority tasks;

Solution:

- K8s + Volcano schedules all workloads;
- queue dynamic resource sharing, DRF and preemption strategy;

Benefit:

- Smooth switching of big data platform from **Yarn to K8s**;
- Cloud-native DAP platform serving **17 countries/regions**, **1,100 users**, with an annual growth rate of **8.1%**;
- DAP platform operating projects **450+**;

VOLCANO
CLOUD NATIVE COMPUTING FOUNDATION
THE LINUX FOUNDATION 开源软件学园

Thanks