Google Cloud Summit
Jakarta

Advanced Scheduling for AI/ML: Orchestrating Ray Applications with KubeRay and Kueue

Ananda Dwi Rahmawati

Cloud & DevOps Engineer
Google Developer Expert - Cloud

Agenda

- What is Ray; What is Kueue?
- GKE Scheduling
- Scheduling Orchestration with Ray and Kueue within GKE
- What’s the Next Step

Introduction - The Challenge within GKE

Orchestrating distributed AI/ML workloads

Resource Fragmentation and Allocation

Workload Preemption and Fairness

Dynamic Autoscaling of Ray Clusters

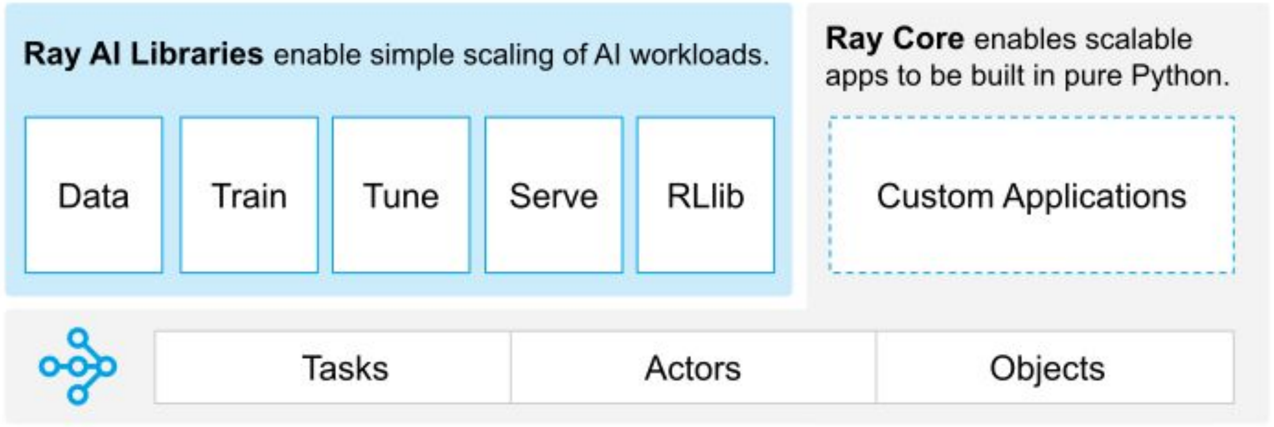
Job Queueing and Backpressure Management

Lack of Job Prioritization

Uncontrolled Cloud Costs

Inconsistent Performance & Predictability

What is Ray?


Ray AI Libraries enable simple scaling of AI workloads.
Data
Train
Tune
Serve
RLlib
Ray Core enables scalable apps to be built in pure Python.
Custom Applications
Tasks
Actors
Objects

A unified framework for scaling AI and Python applications **effortlessly**—from training to tuning to serving—on any infrastructure.

Why Ray on GKE?

1 Simplified Distributed AI

Abstracts away the complexities of distributed programming

2 Scalability

Dynamically scale on GKE nodes, seamlessly leveraging available CPU and GPU resources.

3 Unified Ecosystem

Provides libraries for common AI tasks

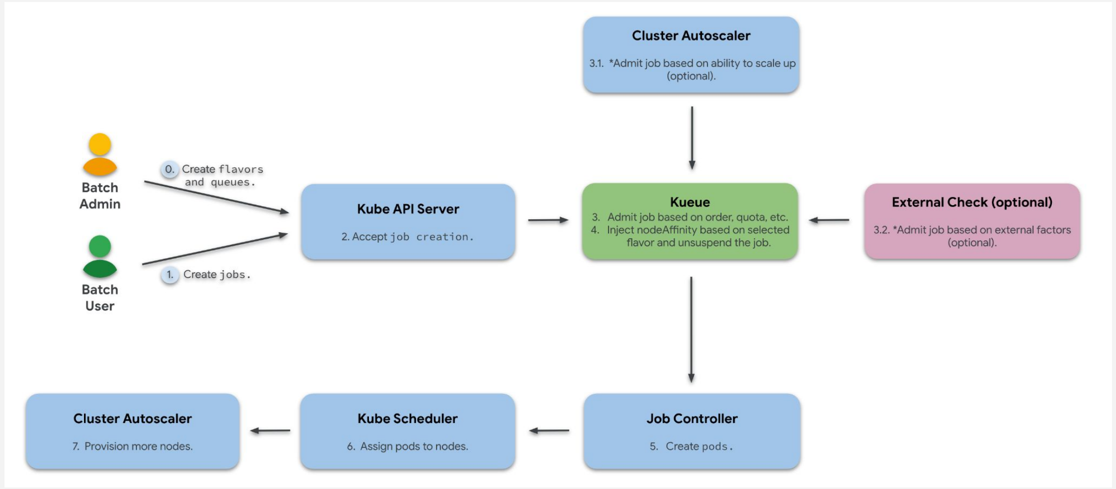
Ray Key Features

Designed to deploy and manage ML models and business logic as production services

- ✓ **Dynamic Batching:** Automatically groups incoming requests to maximize GPU utilization, reducing latency and increasing throughput.
- ✓ **Model Composition:** Easily chain multiple models or pre/post-processing steps into a single, deployable service.
- ✓ **Auto-Scaling:** Automatically scales inference replicas based on real-time traffic and latency metrics.
- ✓ **Traffic Splitting & A/B Testing:** Seamlessly route traffic to different model versions for canary deployments or experimentation.
- ✓ **Python-Native:** Leverage your existing Python code and AI frameworks (TensorFlow, PyTorch, Scikit-learn).

Kueue
and the Challenges it solves

- In shared Kubernetes clusters, AI/ML jobs often compete for limited, expensive resources (like GPUs).
- Without proper queueing, jobs can get stuck, starve, or lead to inefficient resource allocation.
- Ensuring **fair** access and **optimal** utilization across multiple teams or users.


Cluster Autoscaler
3.1. *Admit job based on ability to scale up (optional).
Batch Admin
0. Create flavors and queues.
Batch User
1. Create jobs.
Kube API Server
2. Accept job creation.
Kueue
3. Admit job based on order, quota, etc.
4. Inject nodeAffinity based on selected flavor and unsuspend the job.
External Check (optional)
3.2. *Admit job based on external factors (optional).
Cluster Autoscaler
7. Provision more nodes.
Kube Scheduler
6. Assign pods to nodes.
Job Controller
5. Create pods.

Designed to **manage and schedule** batch workloads, especially common in AI/ML training and inference.

Google Cloud

Scheduling Techniques

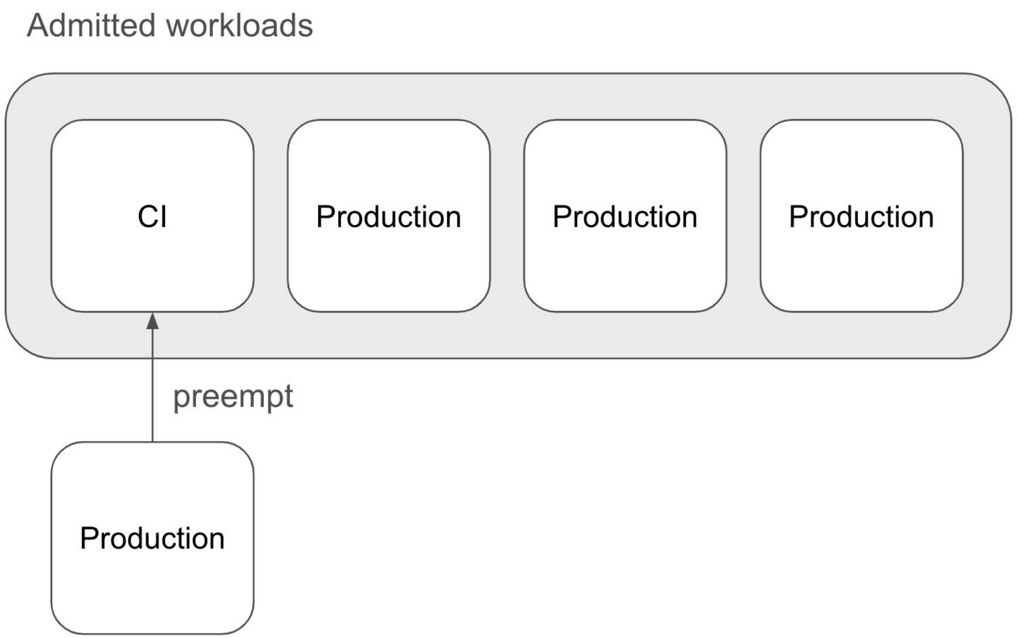
Priority Scheduling

Prioritize AI/ML tasks to ensure production reliability and improve cost efficiency

Gang Scheduling

Orchestrate the execution of tightly coupled AI/ML tasks to maximize resource usage and accelerate training

Priority Scheduling



Challenge: **Prioritizing Critical Batch Workloads**

- **The Scenario:** Our GKE cluster hosts diverse batch workloads, including critical RayJob-based offline inference for production and general CI tests.
- **The Problem:** Finite cluster resources mean these workloads compete, potentially delaying vital production tasks.
- **Our Solution:** Implementing Priority Scheduling to ensure production workloads always take precedence.

Priority Scheduling with Ray and Kueue

Kueue's WorkloadPriorityClass API allows fine-grained prioritization of RayJob and RayCluster resources within your GKE environment

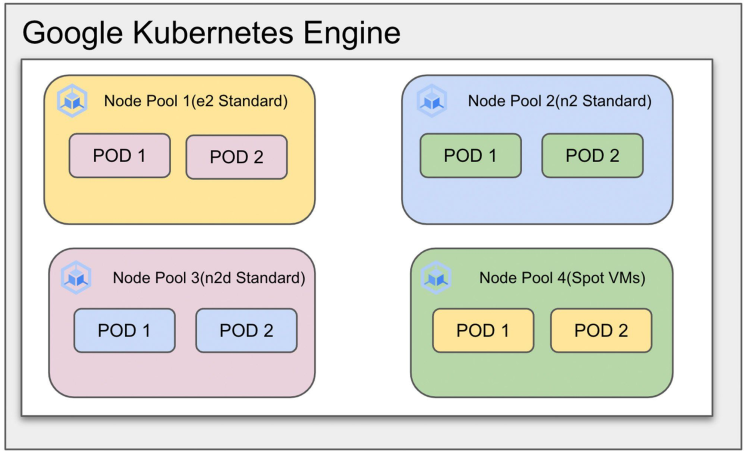
Two Key Impacts of Priority:

- **Queue Order**: Higher-priority workloads are executed earlier within the ClusterQueue
- **Resource Preemption:** When a ClusterQueue lacks sufficient quota, higher-priority incoming workloads can trigger preemption of already admitted, lower-priority workloads

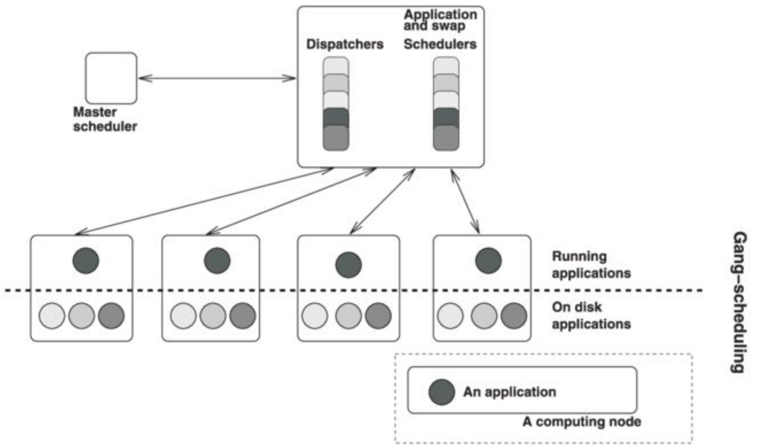
```yaml
---
apiVersion: kueue.x-k8s.io/v1beta1
kind: WorkloadPriorityClass
metadata:
  name: prod-priority
value: 1000
description: "Priority class for prod jobs"
---
apiVersion: kueue.x-k8s.io/v1beta1
kind: WorkloadPriorityClass
metadata:
  name: dev-priority
value: 100
description: "Priority class for development jobs"
```

Gang Scheduling

- **The Goal:** Ensuring that all components of a distributed workload start simultaneously.
- Kueue adopts an **all-or-nothing** approach to workload admission;
- RayJobs and RayClusters are scheduled only when all their required resources are fully available
- To improve resource **efficiency** by preventing scenarios where clusters are partially provisioned and unable to execute tasks



Gang Scheduling with Ray + Kueue on GKE



Essential for **optimizing** resource utilization, especially for **limited** and **expensive** hardware accelerators like GPUs and TPUs in AI/ML workloads.

- Ray job is queued until all requested resources are available
- Kueue issues a ProvisioningRequest to GKE
- GKE's autoscaler provisions required nodes in one step
- Ray Pods are then scheduled together, ensuring synchronized startup

Benefits:

- Prevents **resource waste** (e.g., idle GPUs)
- Improves job **reliability** and training **efficiency**
- Essential for workloads with tight inter-worker coordination

What’s the Next Steps?

- ✓ Explore KubeRay & Kueue Demos
- ✓ Identify & Pilot Key AI/ML Workloads
- ✓ Define Priority Classes & Resource Quotas
- ✓ Monitor & Optimize Performance & Costs

Thank you

@misskecupbung