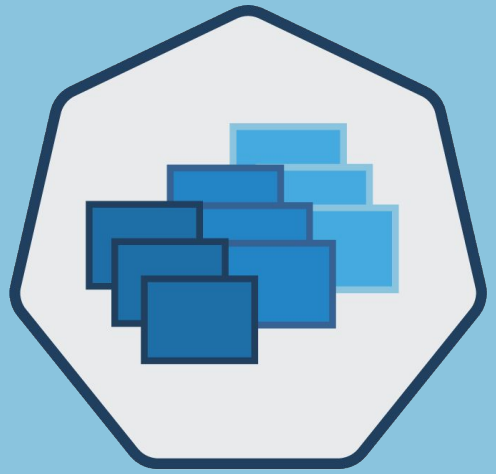


KUBERNETES

BATCH + HPC DAY

EUROPE

Kubernetes batch processing at scale - a scheduling perspective

Lim Haw Jia & Fan Deliang
Bytedance

Agenda

- Batch processing in Bytedance Overview
- Problem with the native scheduler
- The Godel Scheduler
  - Performance features
  - Scheduling capabilities
  - Evaluation
- Future work
- Open source

Batch processing in Bytedance Overview

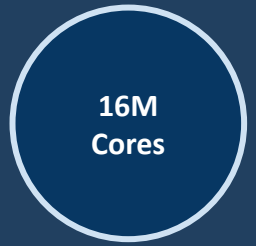
Ecosystem

Business	Recommendation	Search	Advertisement	NLP	CV	.....
Framework					Bytedance ML Framework	
Scheduling		Bytedance Unified Scheduler(Godel Scheduler)				
Hardware	GPU	CPU	ARM	Network	.....	

Large Scale

More than 1M batch jobs running everyday

More than 130M batch job pods run everyday



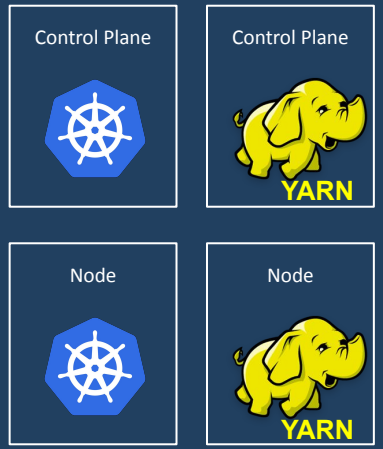
Batch job up to 16M cores

Biggest Cluster manages 20K nodes

Evolution

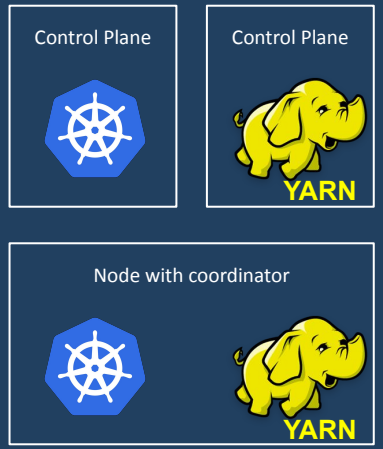
Stage 1: Share Nothing

- Resource fragmentation
- Resource over-provision
- Resource waste during off-peak traffic hours



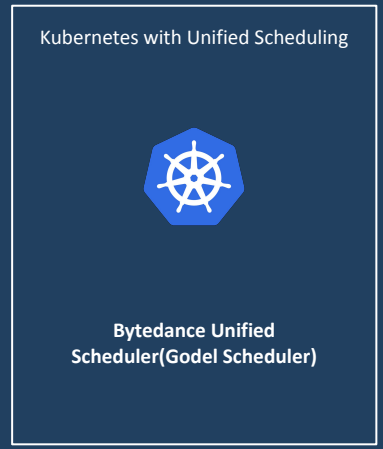
Stage 2: Share Node

- High operational costs
- Undesired resource elasticity



Stage 3: Unified Scheduling

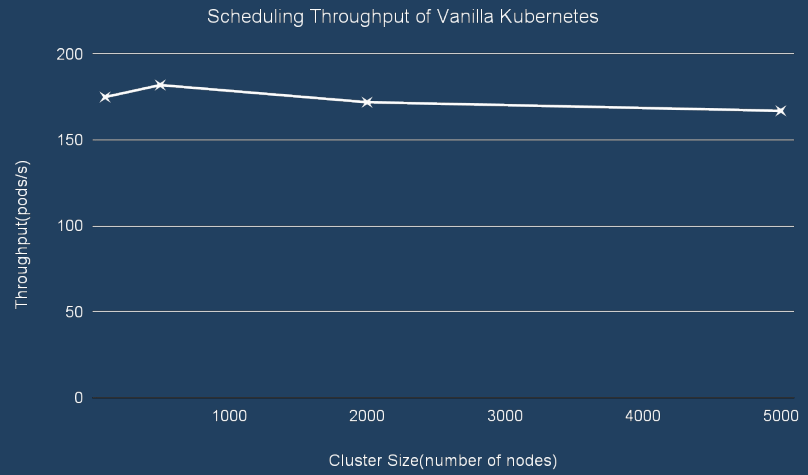
- High resource elasticity and utilization
- High operational efficiency



Problems with the native scheduler

Scalability

- Batch jobs caused the number of pods to increase drastically with high bursts when we reach peak periods
- Kubernetes scheduler does serial processing
- Benchmarks:
  - 160-180 pods per second
  - Scheduling throughput decreases as cluster size increases, due to computational complexity



Capabilities

- Kubernetes was designed for microservices
- Lacks important capabilities for batch processing:
  - Gang scheduling
  - Job level affinity
  - Micro-topology scheduling
  - GPU sharing
- Lacks important capabilities for colocated heterogeneous workloads:
  - Sorting: DRF, priority-based, fair share
  - Preemption
- Extra capabilities
  - Bin-packing

The Godel Scheduler

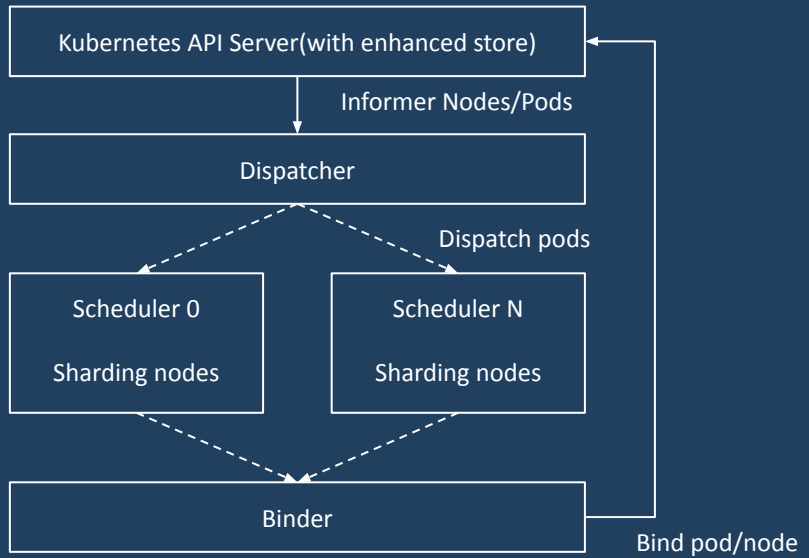
Goals

- High performance scheduler with online/offline scheduling capabilities
- Increase resource utilization
- Unified resource pool with high resource elasticity

Architecture

Key features

- Distributed scheduling system
- Multiple distributed components
- Distributed scheduling with multiple scheduler instances can be scaled horizontally
- Consistency is achieved with optimistic concurrency control



Scheduling framework

Dispatcher

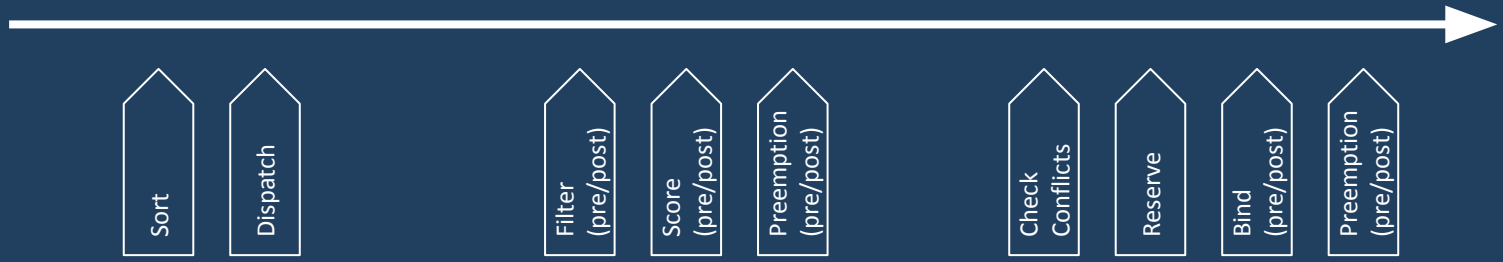
- Translate pods to scheduling unit
- Sort scheduling requests

Scheduler

- Assigns node to pods
- Execute the filter and score stages
- Suggest candidates for preemption

Binder

- Resolve scheduling conflicts
- Executes the reserve, permit and bind stages
- Preempts pods



Performance features

Concurrent scheduling

Godel allows multiple scheduling requests to be processed in parallel, improving the overall scheduling throughput

How it is implemented:

- Multiple scheduler instances with optimistic concurrency control implemented in the binder
- Scheduler instances can be scaled horizontally to increase throughput. However, more schedulers also means more chance of conflicts, thus a balance is needed

Reducing conflicts

- Node partitioning can be used to reduce chances of conflict + reduce time spent on filtering. However, there is a tradeoff with scheduling quality

Result caching

Godel improve the scheduling performance significantly by caching feasible nodes

- All pods in a deployment request share the same template
- A feasible node fitting one pod is likely to be suitable for others in the same deployment

How it is implemented

- Feasible nodes are cached during the filter stage to avoid repeated computations

Scheduling capabilities

Gang Scheduling

Pods are scheduled together, with all or nothing semantics, E.g. distributed learning

How it is implemented:

- Godel scheduler does not schedule pods individually but in scheduling units
- Fills the semantic gap in Kubernetes needed to support gang scheduling

Sorting

More complex queueing semantics are needed for offline workloads, E.g. dominant resource fairness, fair share

How it is implemented:

- The dispatcher stores scheduling requests in a priority queue sorted according to the desired policy
- Dispatches requests to the schedulers in order of priority

Microtopology scheduling

Batch and ML jobs are characterized by high I/O and frequent memory access. Microtopology scheduling, where CPU and memory from the same NUMA node are bound to a container can improve performance.

How it is implemented:

- Godel provides NUMA-aware scheduling with the help of a custom node agent that writes topology information as node annotations

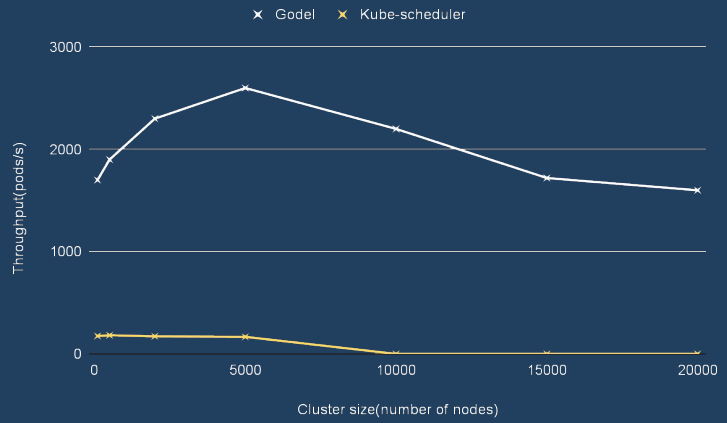
Other capabilities

- GPU sharing
- Job level affinity
- Preemption
- Non native resource scheduling support
- Bin-packing

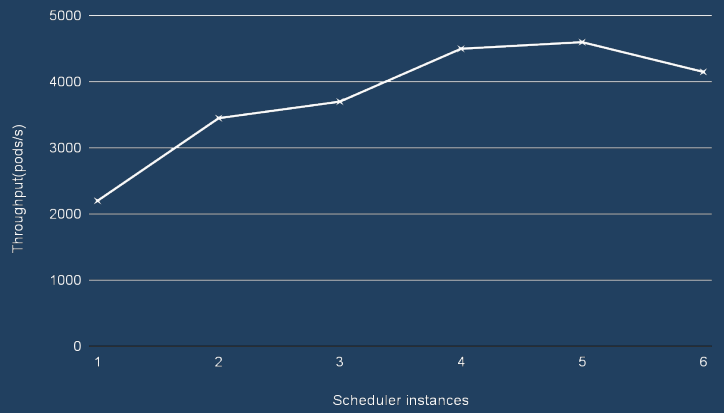
Evaluation

Benchmarks

Compare Kubernetes with Godel with a single scheduler



Throughput under varied instances in a 10k nodes cluster



Note: 40 Debian x86_64 servers, each of which contains 256 logical CPUs, 2TB memory, and 7TB SSD storage

Results

Overall CPU resource utilization up to 60%

Scheduling throughput of 5000 pods/s on a single cluster

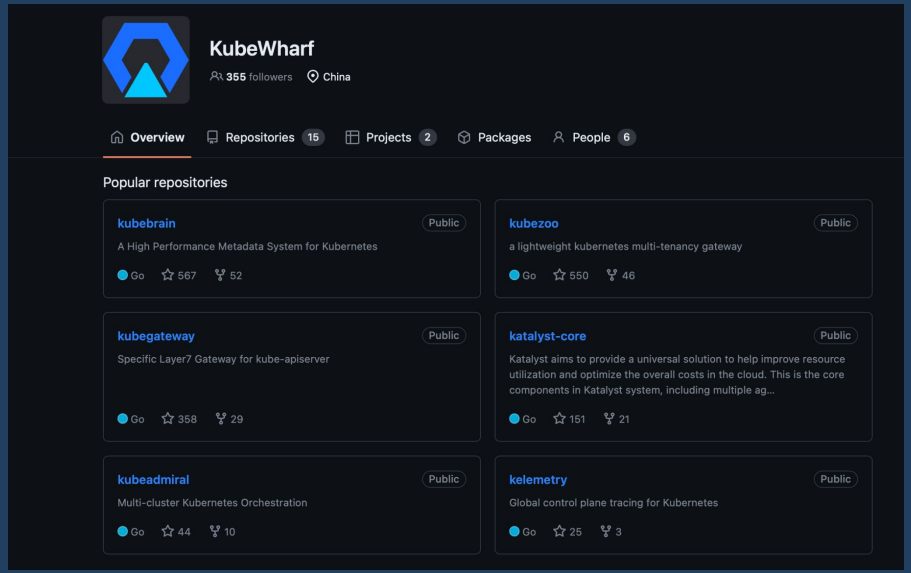
Millions CPU cores elasticity between online/offline service within several minutes

Future work

- Currently, transitory stages used by the dispatcher, scheduler, and binder are persisted in ETCD like store (via API Server). We are working on using an in-memory cache for transitory states, which is expected to scale more than ETCD and nearly double through-put to 10,000 pods/s
- We are actively working on and deploying Godel rescheduler in production. Rescheduler is to watch on running pods and take preemptive actions to reduce fragmentation and resource contention, there by achieving higher QoS for critical workloads
- We are also developing the Godel explainer and Godel simulator to improve the explainability of scheduling decisions.

Open Source

- Expected date: 2023 Q4
- Will be open-sourced under Kubewharf - https://github.com/kubewharf
- Stay tuned!



KUBERNETES
BATCH + HPC DAY
EUROPE

Thank you!