DBS

Live more,
Bank less

Decoding Kubernetes Batch Schedulers

Unraveling the Differences for Optimal Workload Management

Technology & Operations – Data Platform

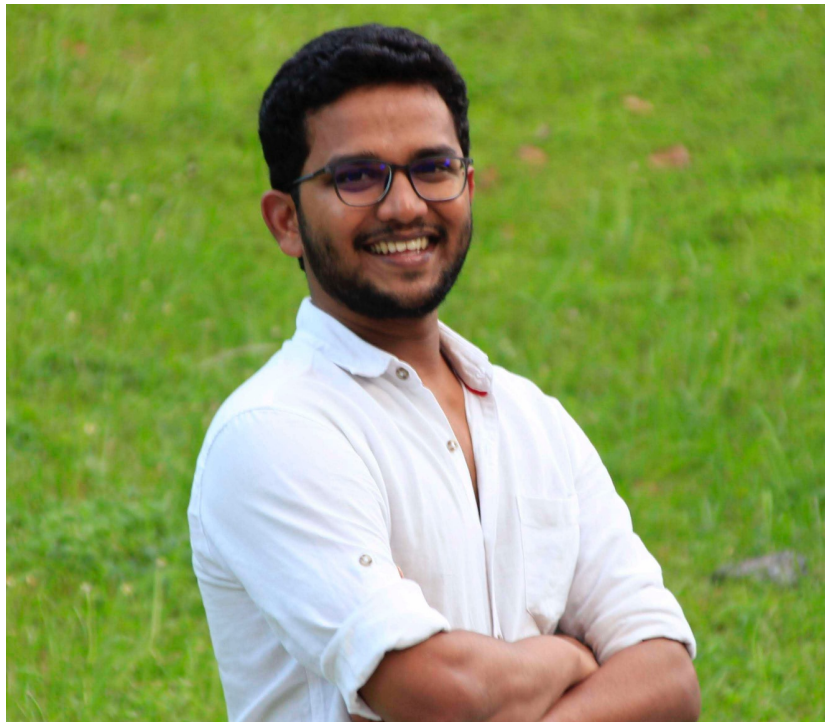
Community Overcode 2023, ASF Conference

11 Oct 2023

Introduction

Naganarasimha Garla

- Senior Vice President, Data Platform
- Senior Principal Engineer
- Apache Hadoop PMC



Harish Kumar Malaga

- Vice President, Data Platform
- Principal Engineer
- Trino Contributor

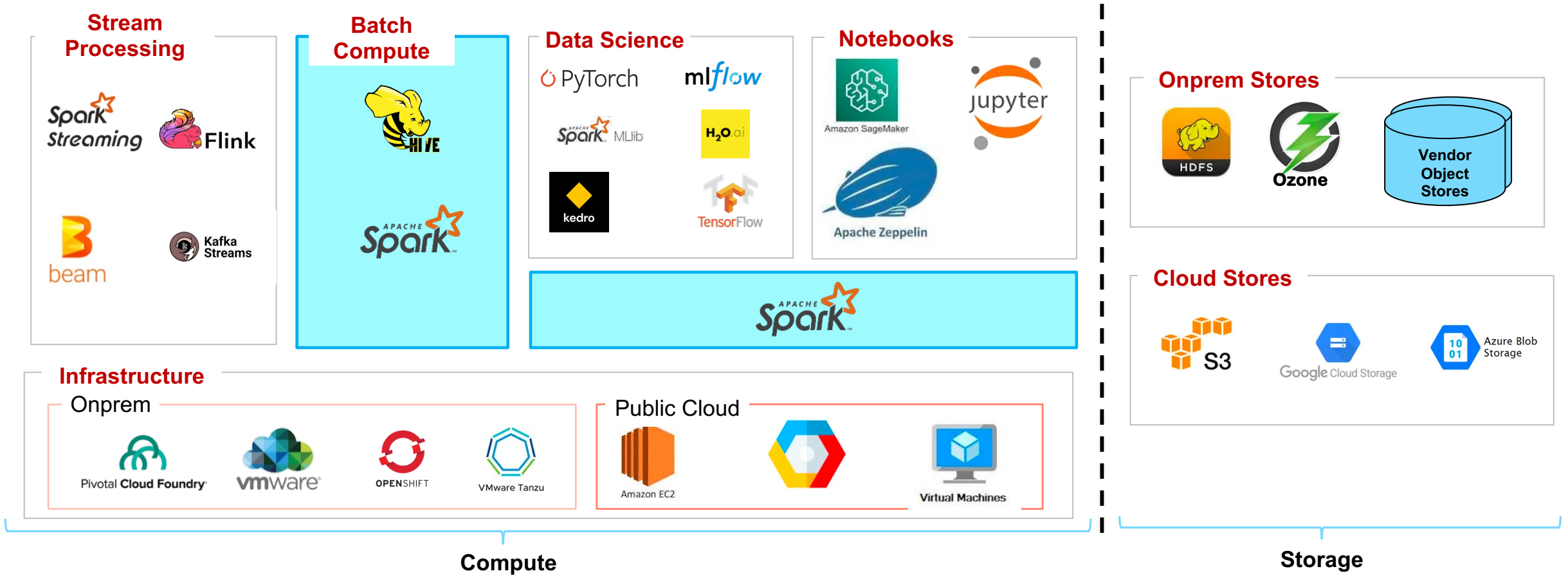
DBS

Agenda

1. **Enterprise ecosystem and motivation for Kubernetes (K8s)**
2. K8s Scheduling
3. Desired Batch Scheduling Features
4. Batch Schedulers on the Table
5. What We Chose and Why
6. Gaps and Enterprise Requirements

DBS

Enterprise Data Ecosystem



- Majority of the compute (*and thus its infra*) is on Spark
- Generally, these Spark batch computes are run on bare metal or VMs in all environments
- Most of the tech stack in the ecosystem can be containerised and managed by a Container orchestrator

DBS

Motivation To Move Batch Workloads To K8s

Cost Optimisation

- Reduced operation costs
- Unified tech stack for both on-prem and cloud

Agility

- Hybrid app support
- Better resource isolation
- Dependency management

Containerisation

- Portability of the apps
- Better resource isolation
- Dependency management

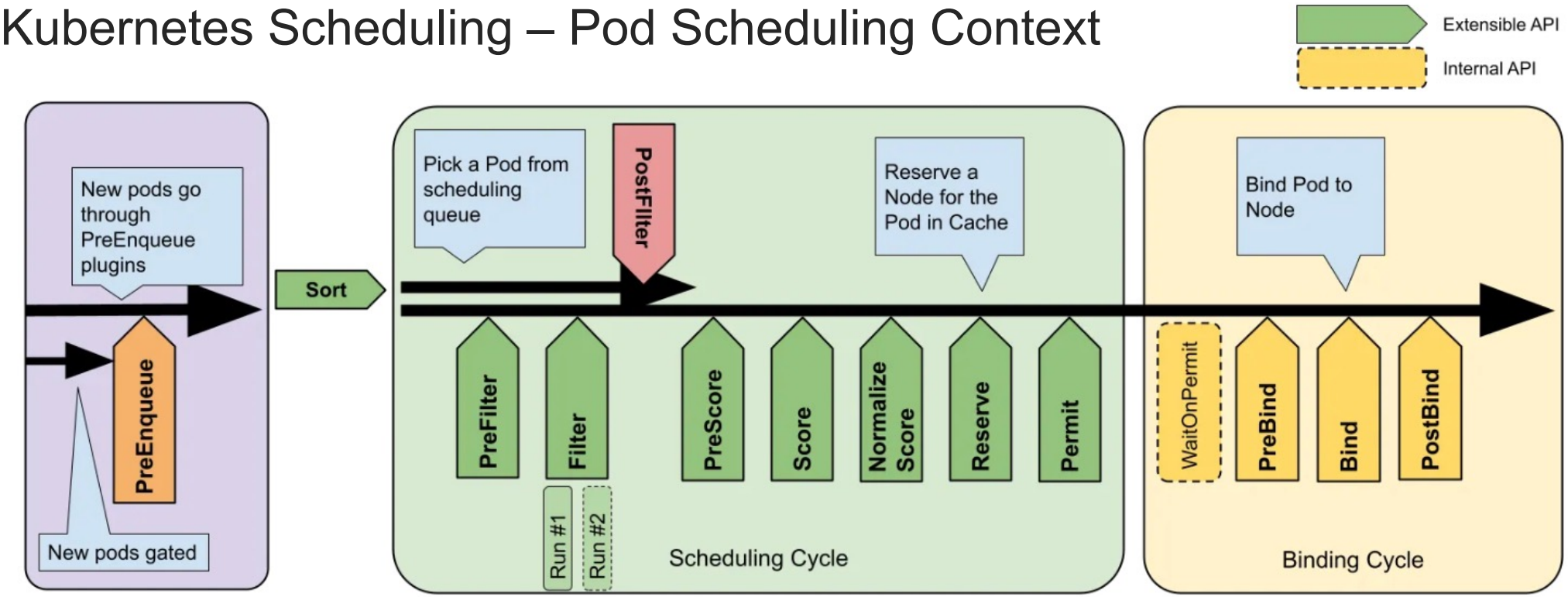
DBS

Agenda

1. Enterprise ecosystem and motivation for Kubernetes (K8s)
2. **K8s Scheduling**
3. Desired Batch Scheduling Features
4. Batch Schedulers on the Table
5. What We Chose and Why
6. Gaps and Enterprise Requirements

DBS

Kubernetes Scheduling – Pod Scheduling Context



- Kubernetes supports Custom scheduler
- **Default scheduler:** Extensible Framework with 3 stages
  - **Pod Queuing:** Parking pods for certain conditions to be met
  - **Scheduling:** Filter – Pod filtering
    PostFilter – Mapping the ideal Node
  - **Binding Cycle:** Launching and bookkeeping

DBS

Kubernetes Scheduling

Scheduling features

- Supports multiple resource types
- Supports labels, taints & tolerations
- Supports bin packing
- Supports dynamic resource allocation
- Supports pod priority & preemption
- Self healing – Node pressure eviction
- Supports Affinity & Anti-affinity

Supports multiple deployment types enabling multiple types of applications

- **Limits/Quota**
  - Resource Quotas are not part of the Default Scheduler
  - Namespace level **Hard enforcement** by rejection during submission
  - Lack of support for resource sharing between jobs, queues, and namespaces
- **No First-class Application Concept**
  - Lack of fine-grained lifecycle management
  - Lack of support for frameworks like Tensorflow, Pytorch, Mxnet
- **Scheduling**
  - Single Queue servicing the entire cluster
  - Single sorting algorithm – FiFO
- **Scale and Performance**
  - Cannot scale # of nodes (~7,500 nodes)
  - Comparatively slower for large batch workloads demands

DBS

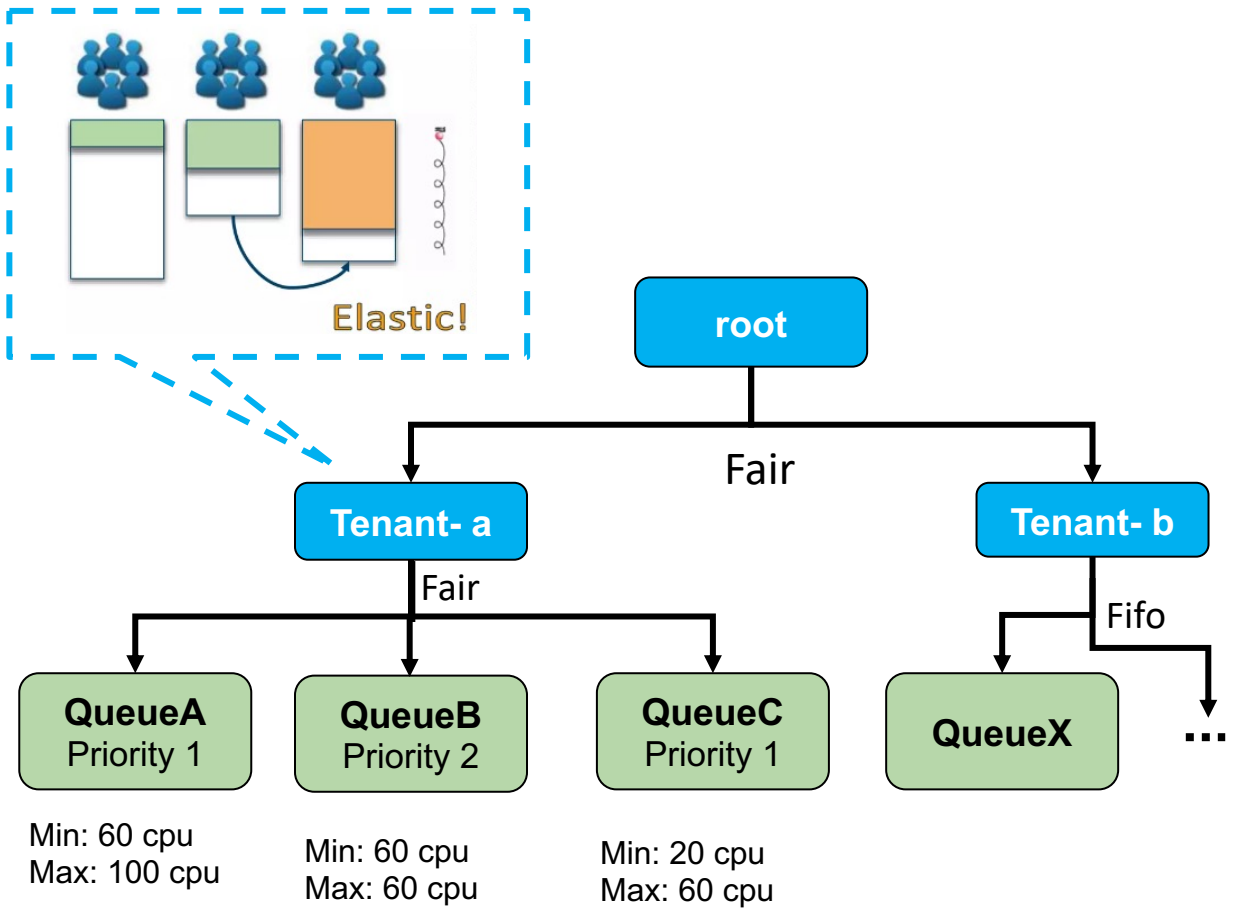
Agenda

1. Enterprise ecosystem and motivation for Kubernetes (K8s)
2. K8s Scheduling
3. **Desired Batch Scheduling Features**
4. Batch Schedulers on the Table
5. What We Chose and Why
6. Gaps and Enterprise Requirements

DBS

Batch Scheduling Features

- **Better Capacity Planning**
  - Define capacity in terms of queues/pools with quotas and limits
  - Define queue hierarchy
  - User-based quota
  - Preemption
- **Advanced Resource Scheduling Features With Application Awareness**
  - Ordering policies: App, Node, Requests
  - Gang scheduling
  - App/Job priority
  - Resource reservation
  - High throughput
  - Topology-based scheduling
  - Reclaim and backfill



DBS

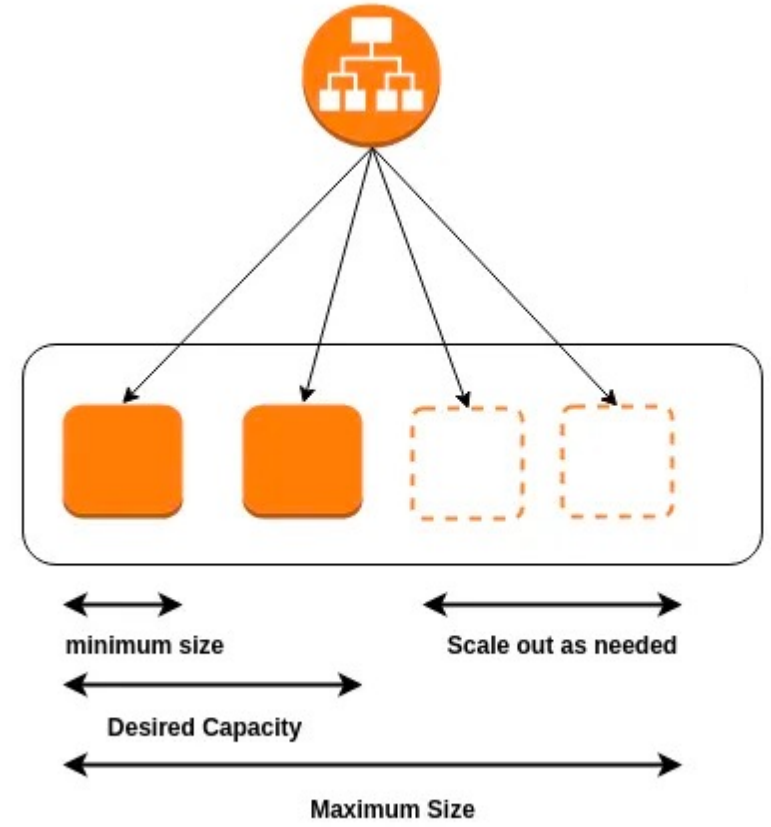
Batch Scheduling Features

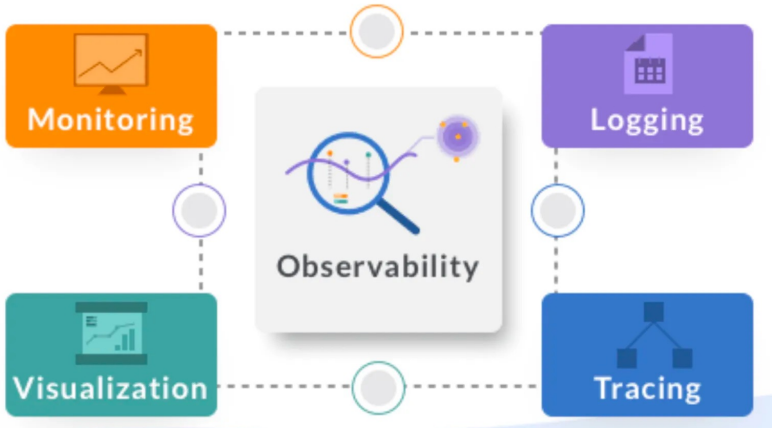
Cloud requirements

- Scale based on the queue's capacity not based on unscheduled pods
- Node Sorting (*Bin packing*) to avoid unwanted node provisioning
- Provisioning pods for an app in the same zone
- Ability to manage budgets

Observability and Troubleshooting

- Centralized place to monitor current state
- Trend analysis of utilization
- Troubleshooting and narrowing down to the troublesome app/user/queue/queue hierarchy
- Admin CLIs to troubleshoot





DBS

Agenda

1. Enterprise ecosystem and motivation for Kubernetes (K8s)
2. K8s Scheduling
3. Desired Batch Scheduling Features
4. **Batch Schedulers on the Table**
5. What We Chose and Why
6. Gaps and Enterprise Requirements

DBS

Batch Schedulers Evaluated

DBS

Volcano

Key dates	Created in March 2019 Accepted by CNCF as Incubator project in April 2022
Maintained by	CNCF
Number of contributors	185
Original creator	Huawei
Latest release	1.8.0 (Aug 2023)

Major Adopters:

50 +

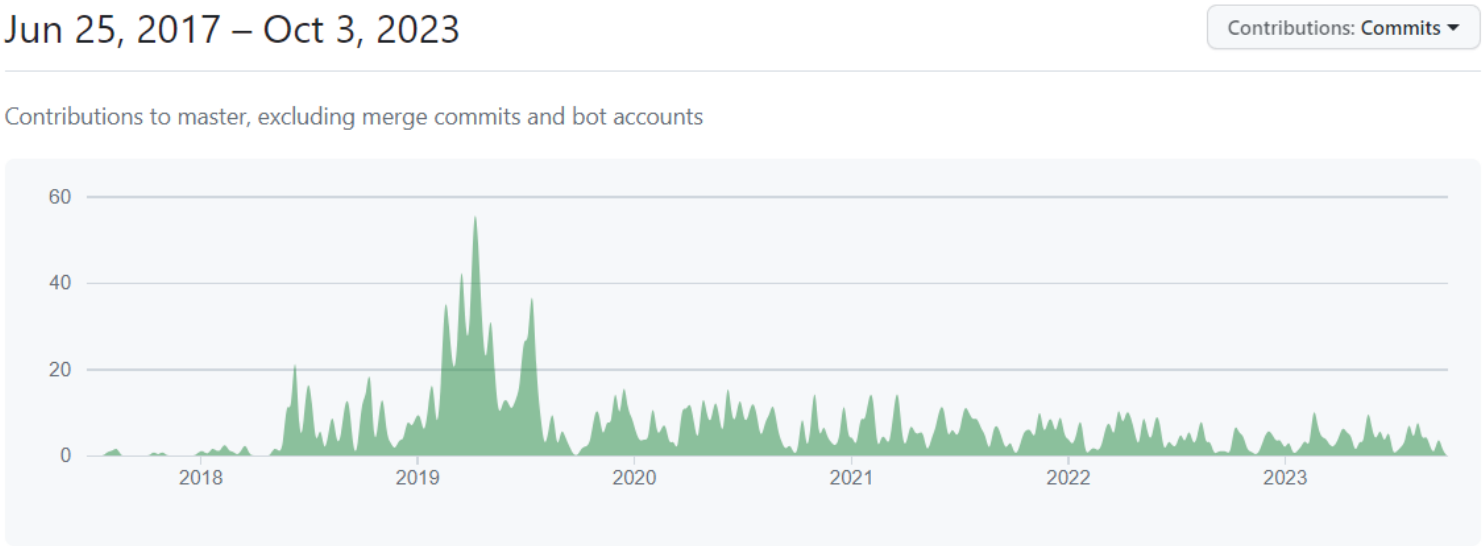
HUAWEI

Tencent 腾讯

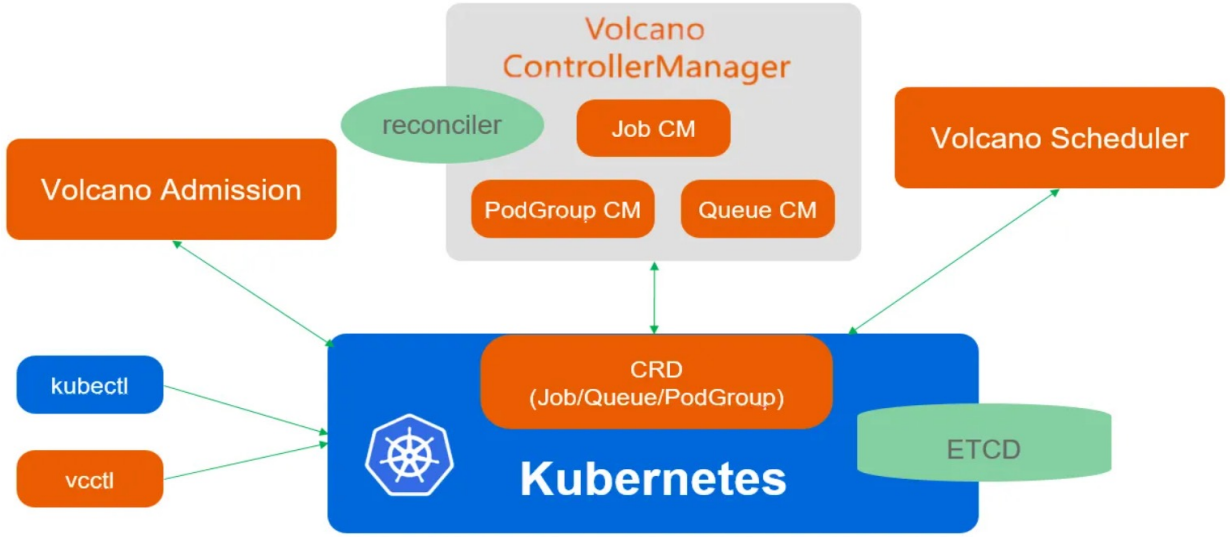
Baidu 百度
百度一下 你就知道

DBS

Contribution trend:

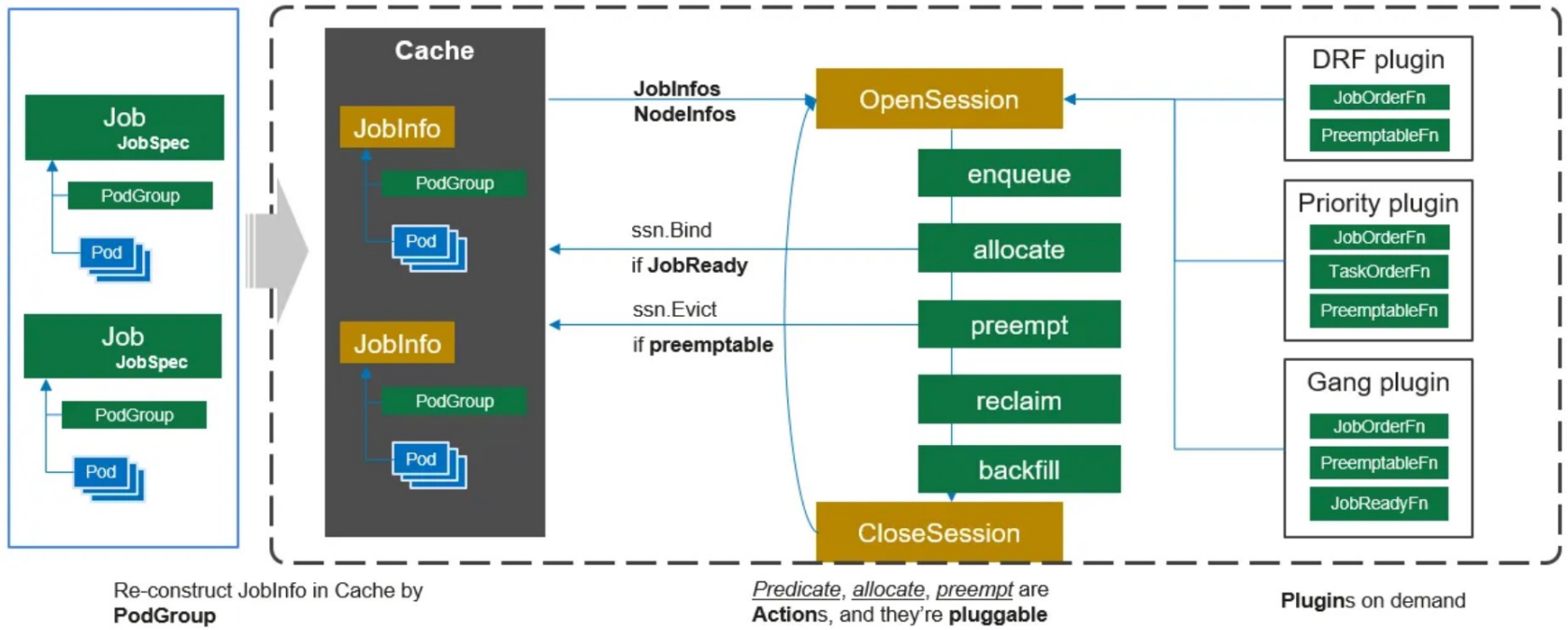


Volcano Architecture



- **Scheduler:** Volcano Scheduler schedules jobs to the most suitable node based on actions and plug-ins. Volcano supplements Kubernetes to support multiple scheduling algorithms for jobs.
- **Controller Manager:** Volcano CMs manage the lifecycle of Custom Resource Definitions (CRDs). You can use the Queue CM, PodGroup CM, and VCJob CM
- **Admission:** Volcano Admission is responsible for the CRD API validation
- **Vcctl:** Volcano vcctl is the command line client for Volcano

Volcano Architecture



- Watches for and caches the jobs submitted by the client.
- Opens a session periodically. A scheduling cycle begins
- Sends jobs that are not scheduled in the cache to the to-be-scheduled queue in the session.
- Traverses all jobs to be scheduled. Executes enqueue, allocate, preempt, reclaim, and backfill actions in the order they are defined, and finds the most suitable node for each job. Binds the job to the node. The specific algorithm logic executed in the action depends on the implementation of each function in the registered plugins.
- Closes this session.

Volcano

Pros

- Supports diverse scheduling algorithms
  - Gang scheduling
  - Fair-share scheduling
  - Queue scheduling
  - Preemption scheduling
  - Topology-based scheduling
  - Reclaim
  - Backfill
  - Resource reservation
  - Supports to configure plugins and actions to use custom scheduling policies
  - Elastic jobs
- Allows to use mainstream computing frameworks
  - Spark, TensorFlow, PyTorch, Flink, Argo, MindSpore, PaddlePaddle, Open MPI, Horovod, MXNet, Kubeflow, KubeGene, and Cromwell
- Supports Federation in its latest release

Cons

- Architecture extends a particular version of K8s scheduler and completely replaces the default scheduler
  - All new features supported by default by K8s will not be available if scheduled through Volcano
  - All the supported deployment types from K8s are not supported by Volcano, and hence it's not a complete replacement for the default scheduler
- Though they have metrics that can be captured in Prometheus, the web-based toolkit isn't suitable for troubleshooting as the technology isn't mature enough

Armada

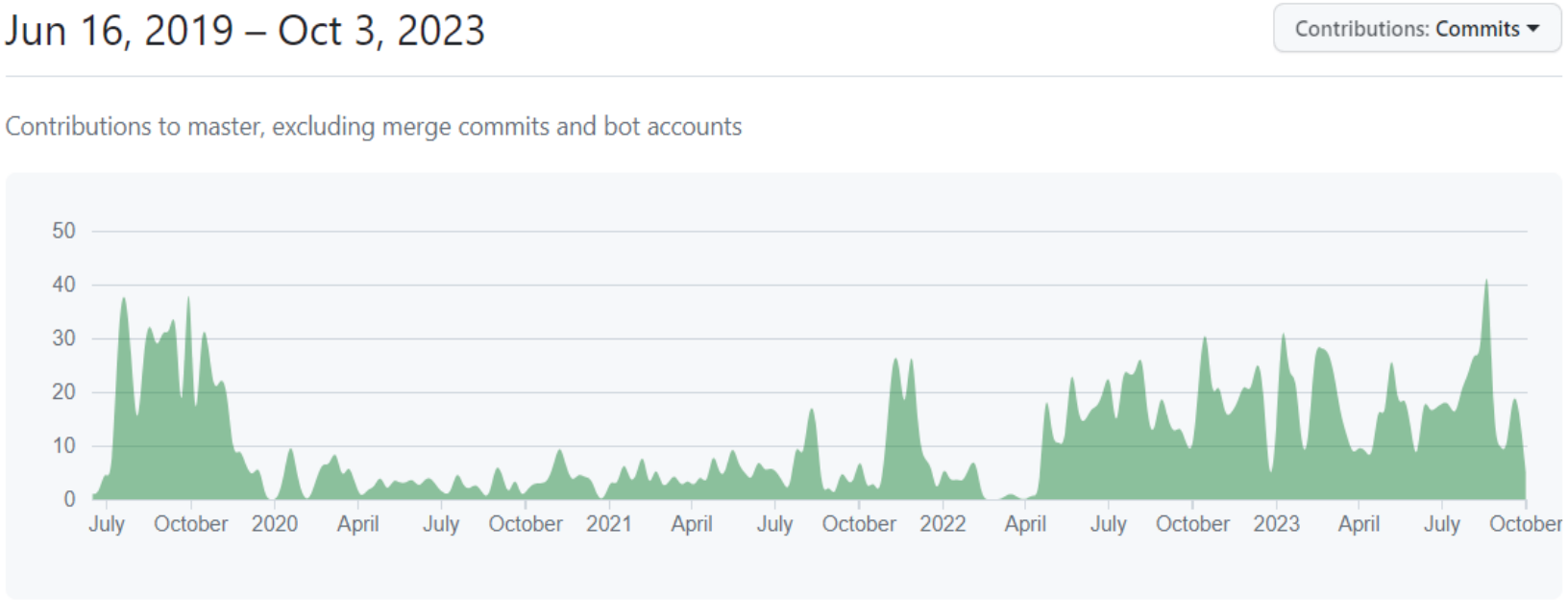
Key dates	Created in October 2019 CNCF Sandbox in April 2022
Maintained by	CNCF
Number of contributors	75
Original creator	G Research
Latest release	0.3.92 (Sep 2023)

Major Adopters:

OpenAI

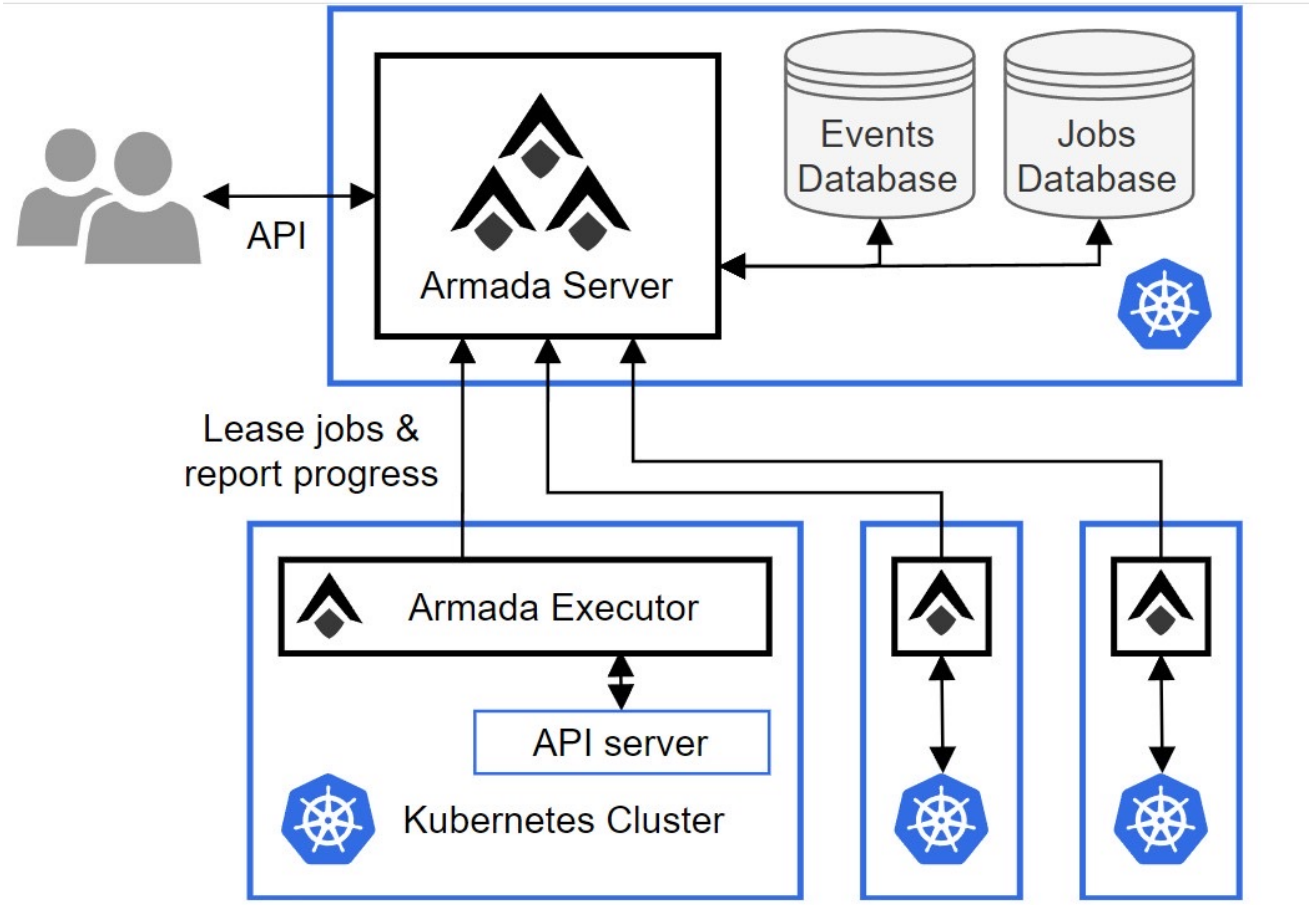
RESEARCH

Contribution trend:



DBS

Armada Architecture



- **Armada server:** Responsible for accepting jobs from users and deciding in what order, and on which Kubernetes cluster the jobs should run. Users submit jobs to the Armada server through the armadactl command-line utility or via a gRPC or REST API.
- **Armada executor**: There is one instance running in each Kubernetes cluster that Armada is connected to. Each Armada executor instance regularly notifies the server of the amount of spare capacity available and requests for jobs to run. Users of Armada never interact with the executor directly.
- All states relating to the Armada server are stored in Redis, which may use replication combined with failover for redundancy. Hence, the Armada server is itself stateless, and is easily replicated by running multiple independent instances. Both the server and the executors are intended to run in Kubernetes pods.

DBS

Armada Overview

Pros

- Primarily designed for
  - Manage compute workloads on tens of thousands of nodes spreading across K8s clusters
  - High throughput, schedules >1,000 pods per second on average
  - Enqueue tens of thousands of jobs over a few seconds.
  - Divides resources fairly between users
  - Provides visibility for users and admins
  - Ensure near-constant uptime

Cons

- Too many layers and components if the workload is not that large
- For advanced scheduling features, it needs to rely on the native scheduler.
- No integration with autoscaling
- Eventual consistent architecture
- Minimal observability

.

DBS

Kueue

Key dates	**Created in October 2021** **CNCF Sandbox in April 2022**
Maintained by	CNCF
Number of contributors	20 (42)
Original creator	Google
Latest release	0.4.1 (Aug 2023)

Major Adopters:

Google Kubernetes Engine

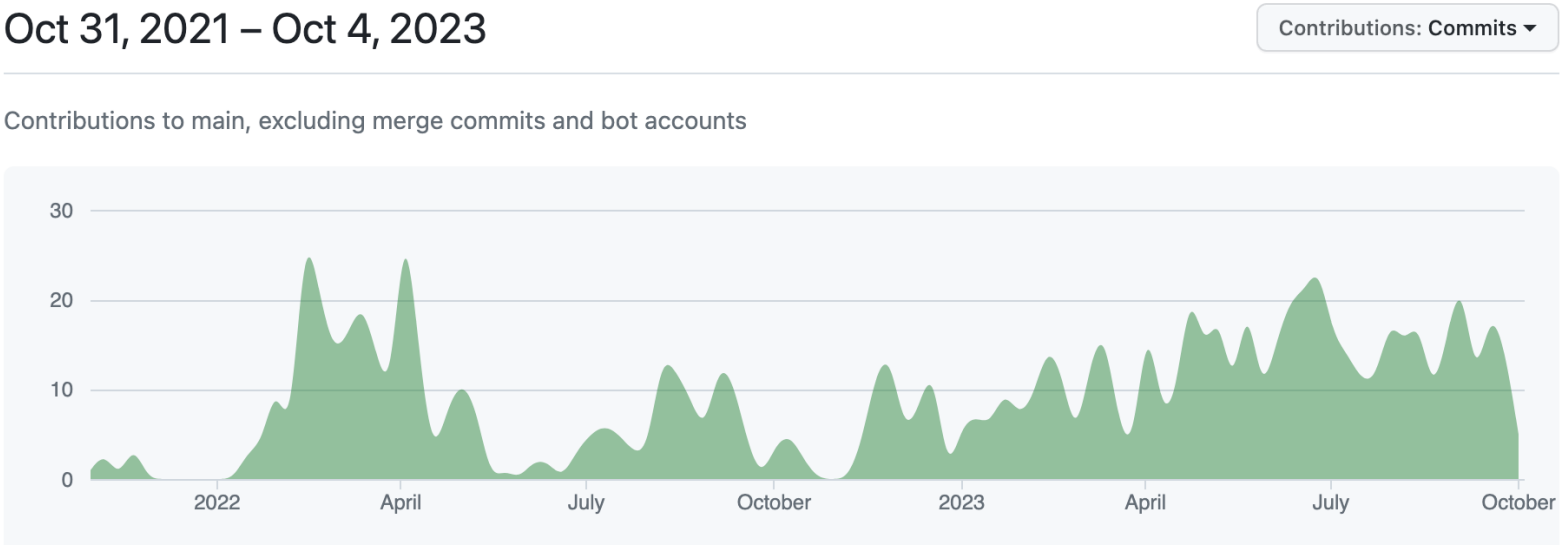
DaoCloud

WATT IQ

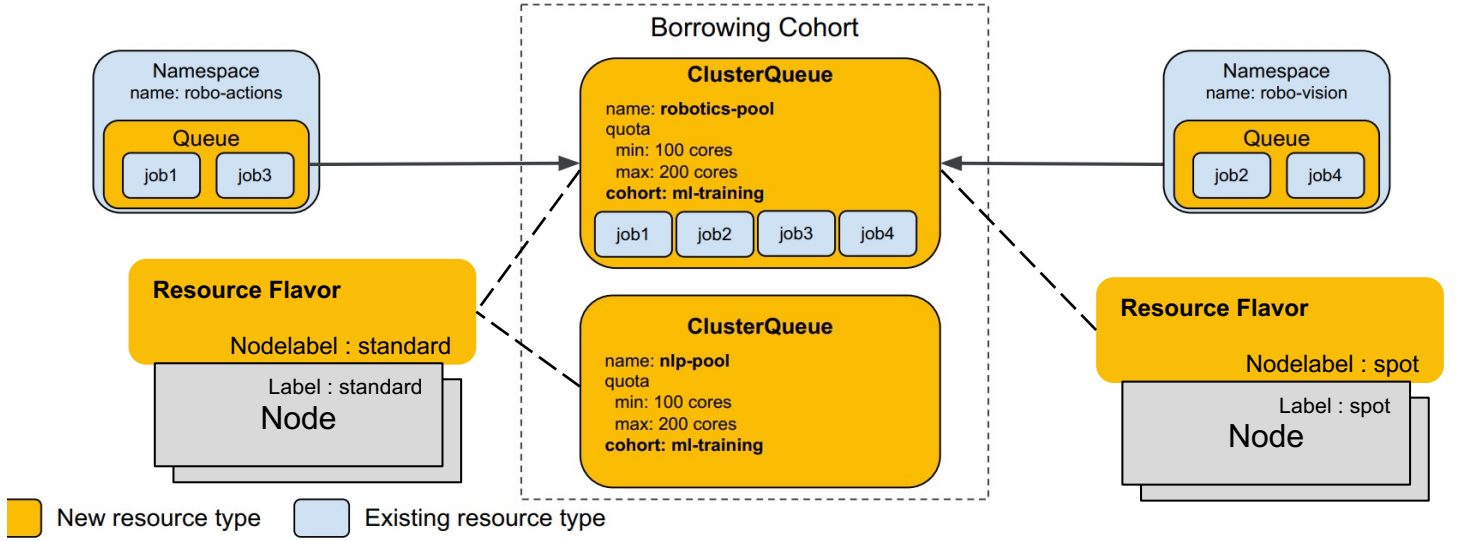
CyberAgent

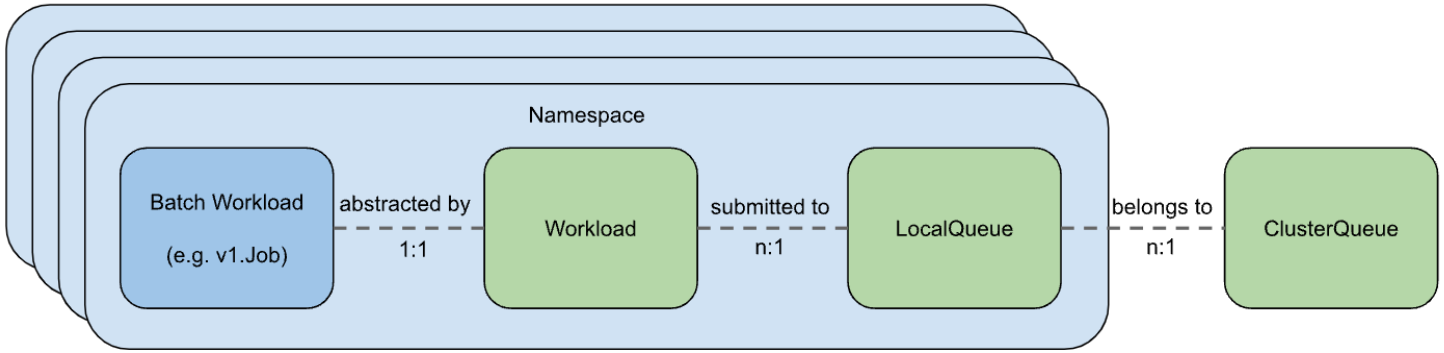
DBS

Contribution trend:



Kueue Resource Model





Key Design Principle: Reuse and extend existing API's to ensure there's no overlapping with existing component's scope

Existing Constructs

Namespace: Canonical tenant abstraction

Job: Computation that runs to conclusion can have one or more similar or different pods

New Resource Constructs:

Queue/LocalQueue: Grouping, and managing closely related tenant jobs

Cluster Queue: Cluster scoped resource that governs the pool of resources, defines usage limits boundaries of fair sharing

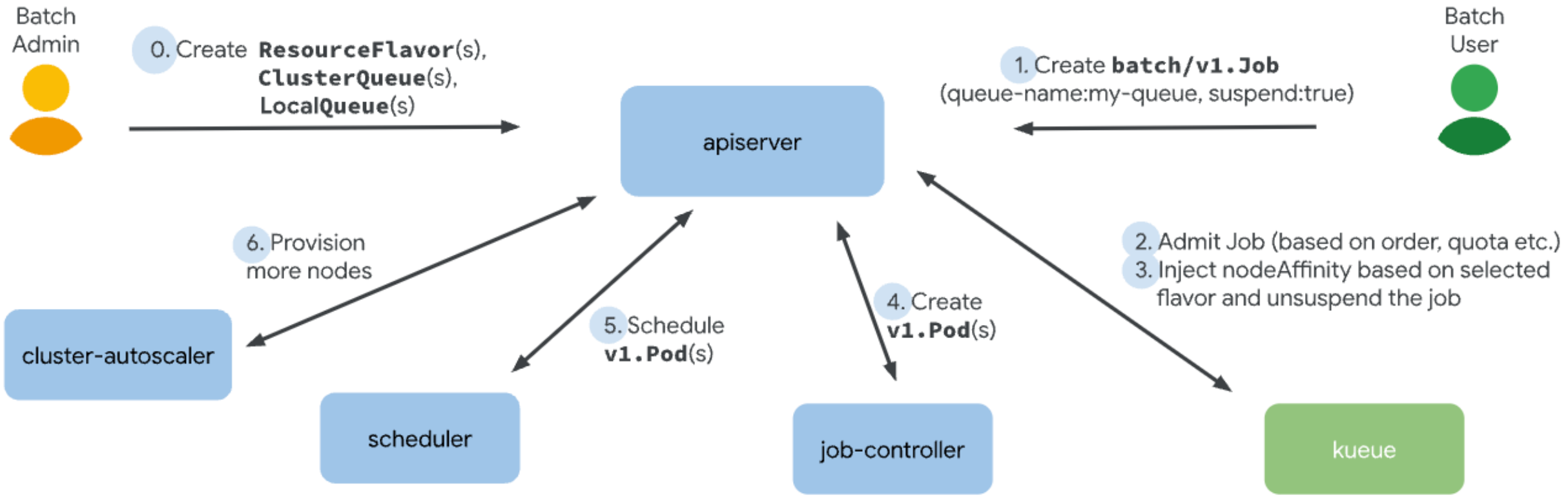
ResourceFlavor: A set of labels that mirrors the labels on the nodes that offer those resources.

Workload: Synonymous to a job that generally comprises one or more pods, and runs to completion.

Cohort : Group of Cluster Queues which can share the resources across the resources

DBS

Kueue Operations



DBS

Kueue overview

- Has been primarily designed to keep both onprem and Cloud in mind, where the latter requires provisions to scale differently for different resources are elastic and heterogenous
- Native to Kubernative and is therefore a complete package. Queuing is just additional controller
- Has been designed without much overlaps with other key components i.e. Pod Scheduling, Autoscaling, Job life cycle mgmt, etc..
- Supports different type of jobs : V1.Job, RayJob, MPI JOB, Flux Minicluster, Python, JobSet and extendable Custom Jobs too.

- Still in experimental stage, not GA ready yet, though it’s supported in GKE
- No gain in Scheduling performance. Cluster >5k will not be addressed with existing approach.
- K8s way of troubleshooting, needs more APIs to make it complete.
- Different perspective to Queue Management that might be complicated for existing YARN users to comprehend.
- Considers all POD’s of an app/job equal.

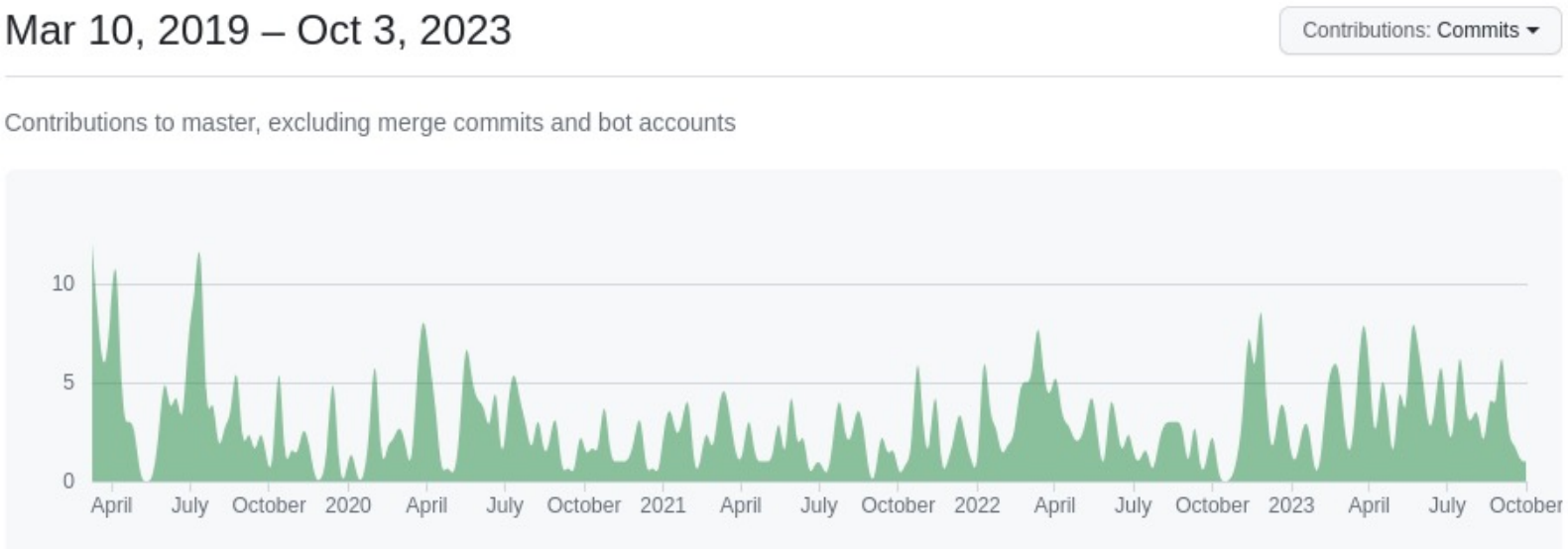
DBS

Yunikorn

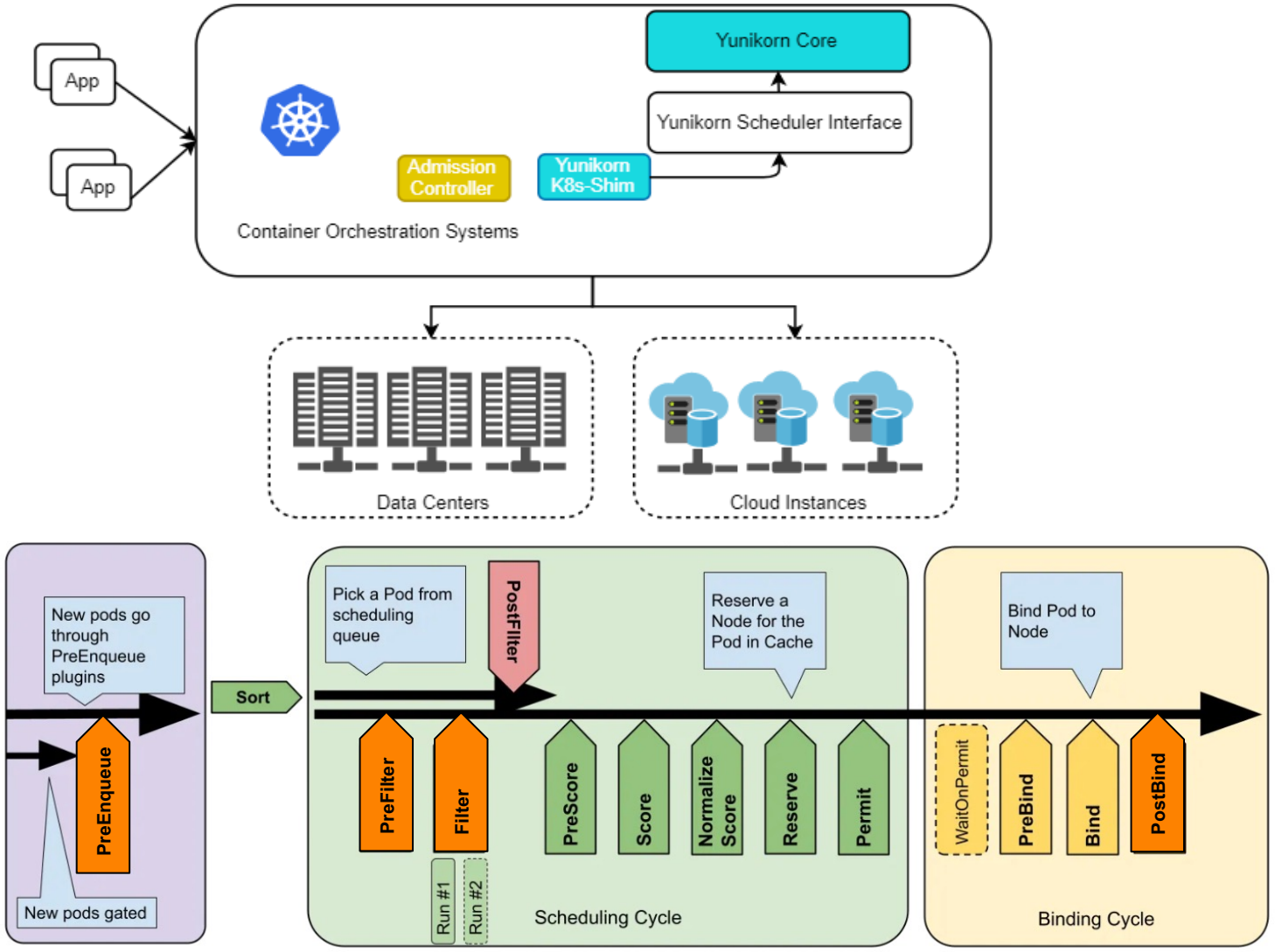
Key dates	Introduced as an Apache Incubation project in 2020 Top Level project in 2022
Maintained by	Apache software foundation
Number of contributors	~40
Original creator	Cloudera and Apple
Latest release	1.3 (Jun 2023)

Major Adopters:

Contribution trend:



Yunikorn Architecture



Architecture

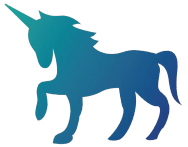
- Yunikorn Scheduler Pod
  - Yunikorn scheduler: Core scheduling decisions
  - Yunikorn Web: For web and ReST interface
- Admission Controller Pod (optional)
  - AdmissionController: Filtering Namespace requests, User mapping or limiting

Scheduler

- Extends K8s scheduler
- 4 stages are mainly extended
  - PreEnqueue
  - PreFilter
  - Filter
  - PostBind

DBS

Yunikorn Overview



Pros

- Predominantly founders were also YARN developers. As such, Yunikorn scheduler feature set almost matches and in certain cases are even optimized than YARN. Highlights include:
  - User/group level resource mapping in Queue hierarchy
  - Sorting at various levels App, Node and Resource weighting
  - Gang scheduling
- Core Design principle has been to K8s default scheduler, which enables it to be applied at a cluster level
- Better troubleshooting with Web UI and rest-based ***"activity dump"***
- Better options to map Namespace to Queues
- Better support for public cloud as it supports Bin packing and enables Queue based scaling support (upcoming version for plugin model).
- Performance is more optimised.

Cons

- Scaling out for larger clusters(>5k) is yet to be addressed.
- Design of native K8s and Yunikorn seems to be diverging, need to see on the sustainability in the long term.
- Requires support for Cloud centric features like budgets, zone specific placement of App's pods needs to be considered
- K8's mapping Yunikorn build needs to be created to ensure all the functionalities are intact.

Agenda

1. Enterprise ecosystem and motivation for Kubernetes (K8s)
2. K8s Scheduling
3. Desired Batch Scheduling Features
4. Batch Schedulers on the Table
5. **What We Chose and Why**
6. Gaps and Enterprise Requirements

DBS

Why and what we choose

- Meets most of the Batch scheduling requirements
  - Able to gracefully to scale out in Cloud
  - Fine grained capacity management
  - Optimised observability and troubleshooting
  - Application aware resource scheduling
- Proven solution at large scale and adopted by large enterprises
- Continuous contributions over a period.
- Active community, which is a mix of organisations
- Not deviating much from K8s scheduling and is easily maintainable in the future
  - Single unified scheduler for all K8s workload
- Sufficient to meet our scale and performant enough to schedule for batch workloads

Yunikorn

DBS

Agenda

1. Enterprise ecosystem and motivation for Kubernetes (K8s)
2. K8s Scheduling
3. Desired Batch Scheduling Features
4. Batch Schedulers on the Table
5. What We Chose and Why
6. **Gaps and Enterprise Requirements**

DBS

Gaps and Enterprise Requirements

We look forward to the following features and aspire to contribute:

- Cloud requirements
  - Tenant based Budget support
  - Deploying a job on a single zone to remove shuffle costs

- There are gaps at Observability when compared to YARN which still needs improvement,
  - Queue's pending apps and reason for not being launched
  - Consumption at user/group level within the Queue.
    - App level metrics :
    - Pending requests
  - Link to the application UI if available
  - Logs (if possible ?)

DBS

Thank You!