KubeCon | CloudNativeCon

North America 2024

KubeCon | CloudNativeCon
North America 2024

LeaderWorkerSet for Distributed Inference

Rupeng Liu(Google)
Abdullah Gharaibeh(Google)

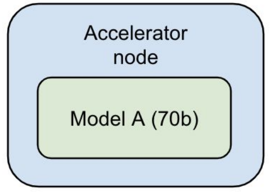
Agenda

Background

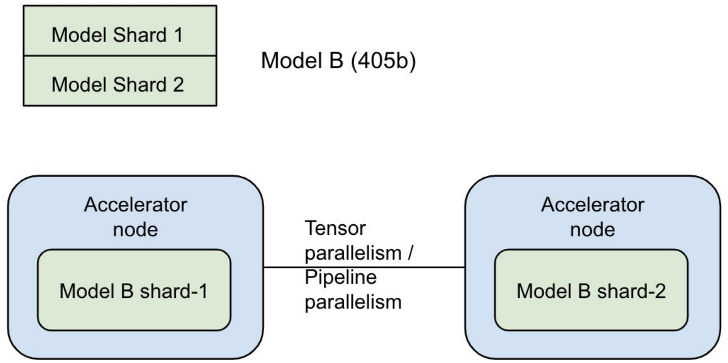
- Large language models are getting bigger, a single accelerator node doesn't have the memory to load the entire model, e.g. Llama3-405B, Mistral-123B etc.

- High Bandwidth Memory (HBM) is important for storing intermediate state and enable higher throughput. Thus we need to shard the model on multiple nodes to efficiently serve the LLM.

Single node serving instance



Multi-node serving instance



Background

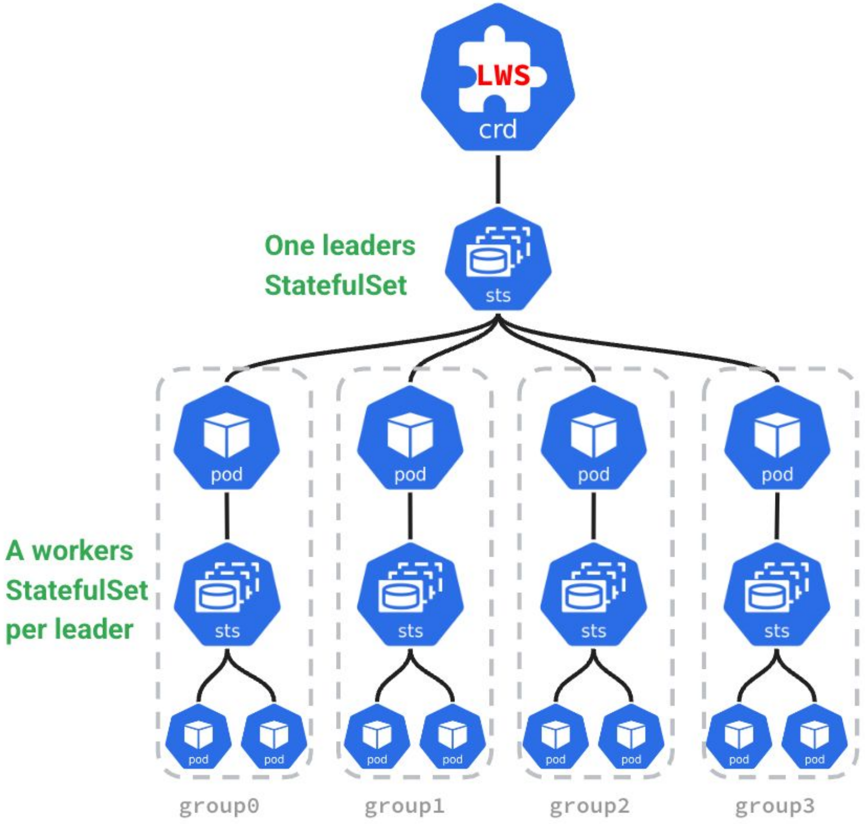
However, there is no automated way to start and replicate a group of pods as a unit.

Solution: a new API named **LeaderWorkerSet**

LeaderWorkerSet Overview

- An API for deploying a group of pods as a unit of replication
  - A group is a “Super Pod”: a single leader and multiple workers
  - Built on top of StatefulSet and supports most of it semantics

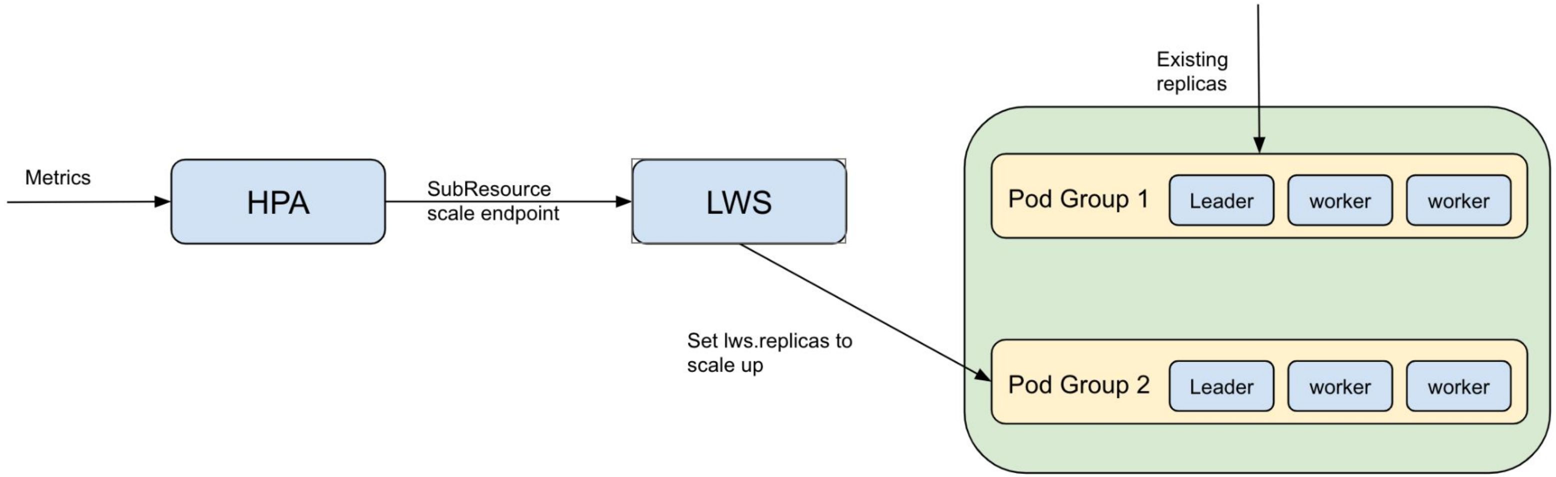
- Features that enables multi-node serving
  - **Pod indices** to assign worker ranks
  - **Dual-template Pods** the leader and the workers have different templates, allows running additional containers on the leader (e.g., the Ray head)
  - **Group startup policy:** create the workers when the leader is ready
  - **Group restart** restarts the pods of a group as a unit when anyone in the group fails
  - **Horizontal-scaling** of groups triggered by HPA
  - **Rolling updates** and rollbacks at the group level
  - **Placement policy** for exclusive 1:1 group to TPU slice placement



Autoscaling

LWS exposes a scale subresource to allow HPA to adjust the number of replicas

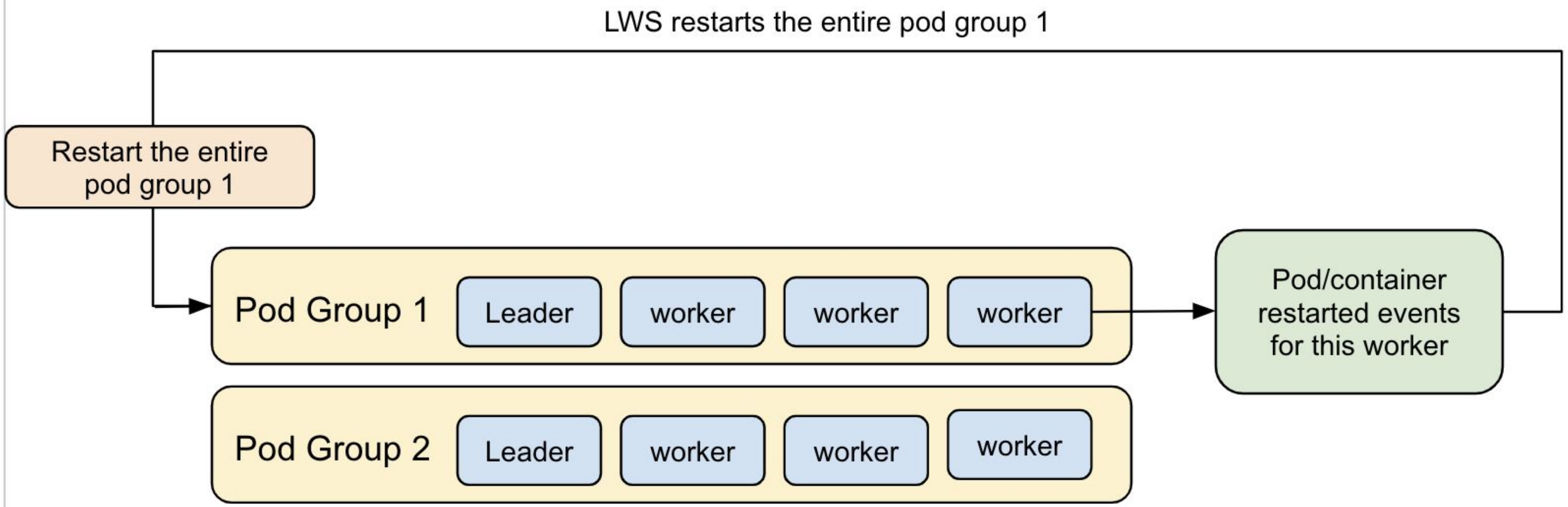
```
subresources:
  status: {}
  scale:
    specReplicasPath: .spec.replicas
    statusReplicasPath: .status.replicas
    labelSelectorPath: .status.hpaPodSelector
```



Failure Handling

Any pod/container restarts in a single serving replica will require the entire replica to be restarted. (group restart)

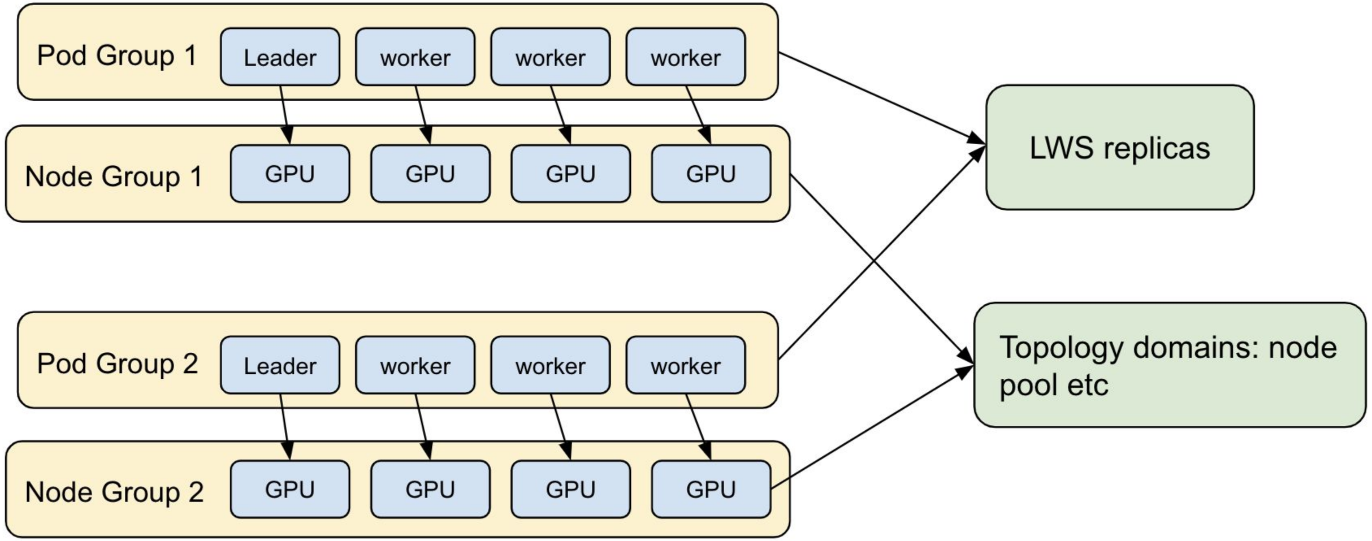
This is because the distributed inference workloads need to be re-initiated when any program on different pod in the same group is restarted.


LWS restarts the entire pod group 1
Restart the entire pod group 1
Pod Group 1
Leader
worker
worker
worker
Pod/container restarted events for this worker
Pod Group 2
Leader
worker
worker
worker

Topology aware scheduling

One pod group/replica will be exclusively scheduled to one topology domain.

- One pod group is a single multi-host serving instance
- A topology domain here is a group of nodes that share the same node label which categorize them to be the same group



Topology aware scheduling

```yaml
apiVersion: leaderworkerset.x-k8s.io/v1
kind: LeaderWorkerSet
metadata:
  annotations:
    leaderworkerset.sigs.k8s.io/exclusive-topology:
google.com/node-pool
  name: multi-node-inference
spec:

...
...
```

```yaml
kind: LeaderWorkerSet
  ...
    affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: leaderworkerset.sigs.k8s.io/group-key
                operator: In
                values:
                - multi-node-inference-0  # pod group name
            topologyKey: google.com/node-pool  # the grouping domai
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions: # exclude the pods of this job
              - key: leaderworkerset.sigs.k8s.io/group-key
                operator: NotIn
                values:
                - multi-node-inference-1
              - key: leaderworkerset.sigs.k8s.io/group-key
                operator: Exist
            topologyKey: tpu-slice-id
```

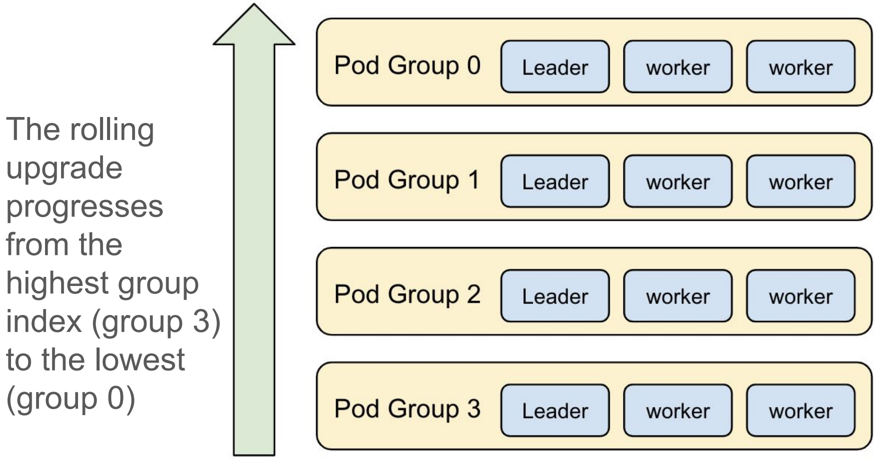
Collocate the pods of a group

Repels all other pods

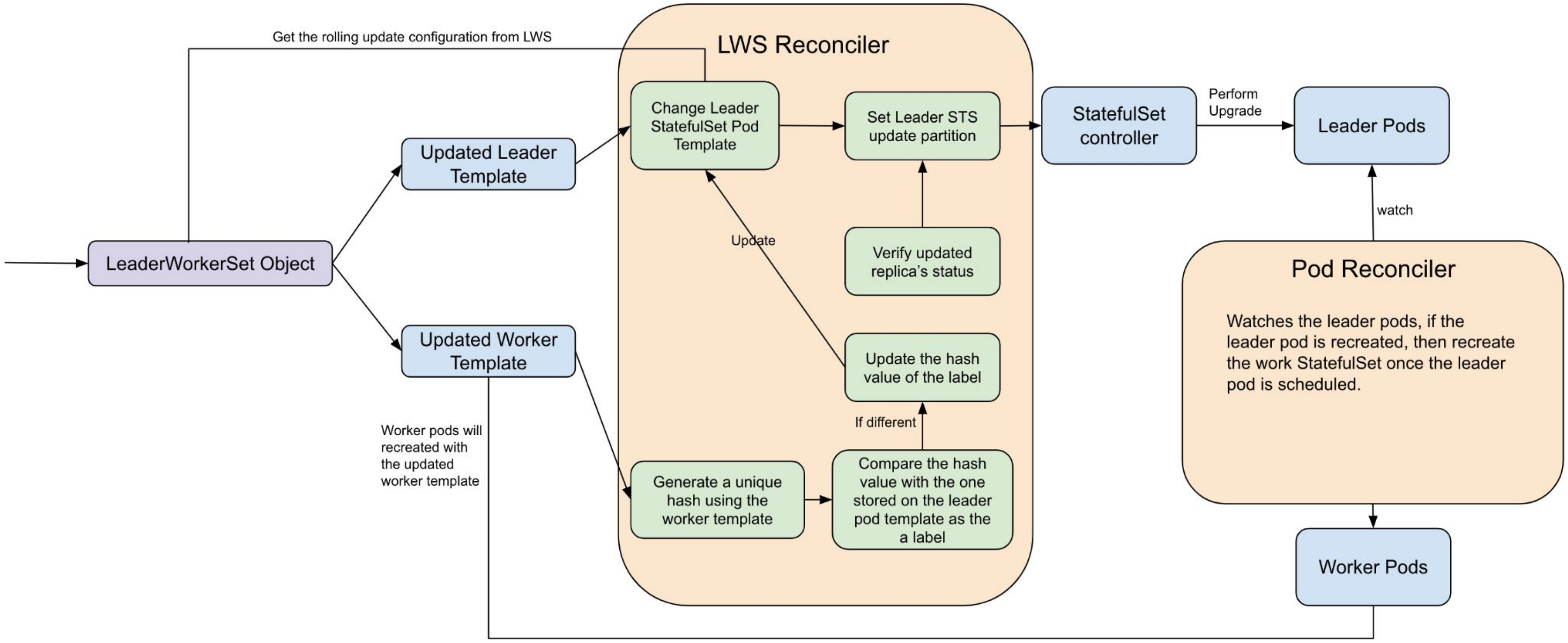
Rolling update

LWS supports rolling update strategy with MaxUnavailable and MaxSurge settings

- Each replica is unit for updating
- **Triggering of the upgrade**: a template hash will be generated for the LeaderWorkerTemplate and some API fields, this hash will be stored as a label on the leader statefulset
- **The upgrade process**: LWS relies on the leader statefulset to manage the process of the upgrade. The upgrade will stop when the upgrade replicas are unhealthy and we use the leader Statefulset's Partition field to control this
- **The restart of the single replica**: The leader statefulset will update the leader pod through recreate and the worker statefulset coupled with the leader pod will be recreated as well through owner reference



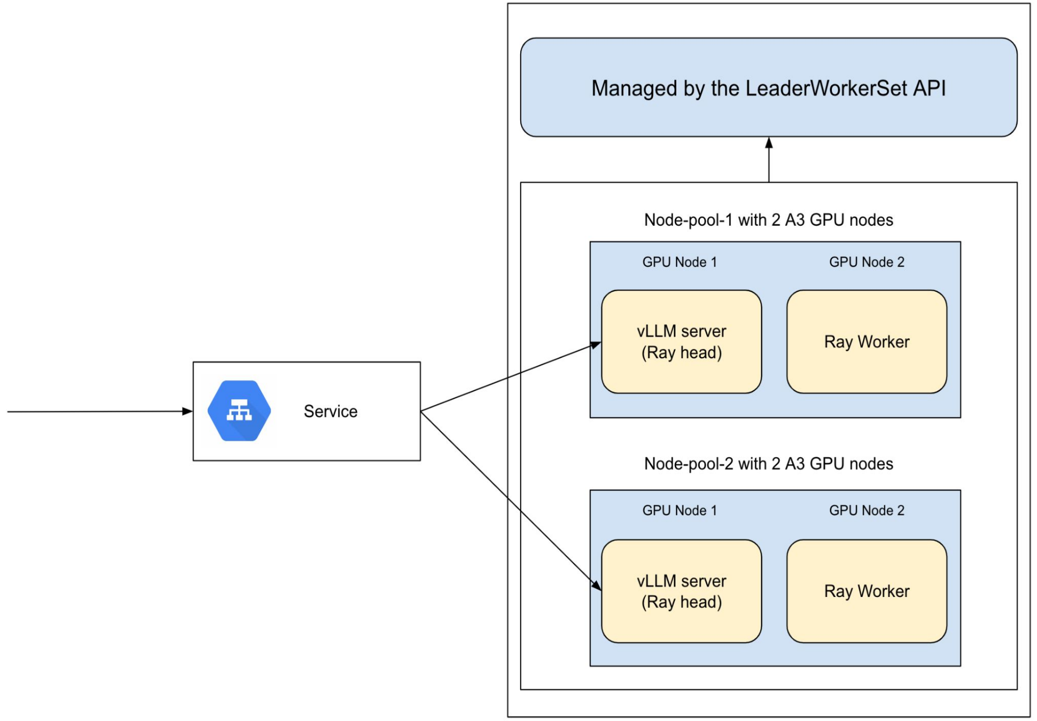
Rolling update


Get the rolling update configuration from LWS
LWS Reconciler
LeaderWorkerSet Object
Updated Leader Template
Updated Worker Template
Change Leader StatefulSet Pod Template
Set Leader STS update partition
Verify updated replica's status
Update
Update the hash value of the label
If different
Generate a unique hash using the worker template
Compare the hash value with the one stored on the leader pod template as the a label
StatefulSet controller
Perform Upgrade
Leader Pods
watch
Pod Reconciler
Watches the leader pods, if the leader pod is recreated, then recreate the work StatefulSet once the leader pod is scheduled.
Worker Pods
Worker pods will recreated with the updated worker template

Multi-node inference on GPU

Infrastructure & runtime setup:

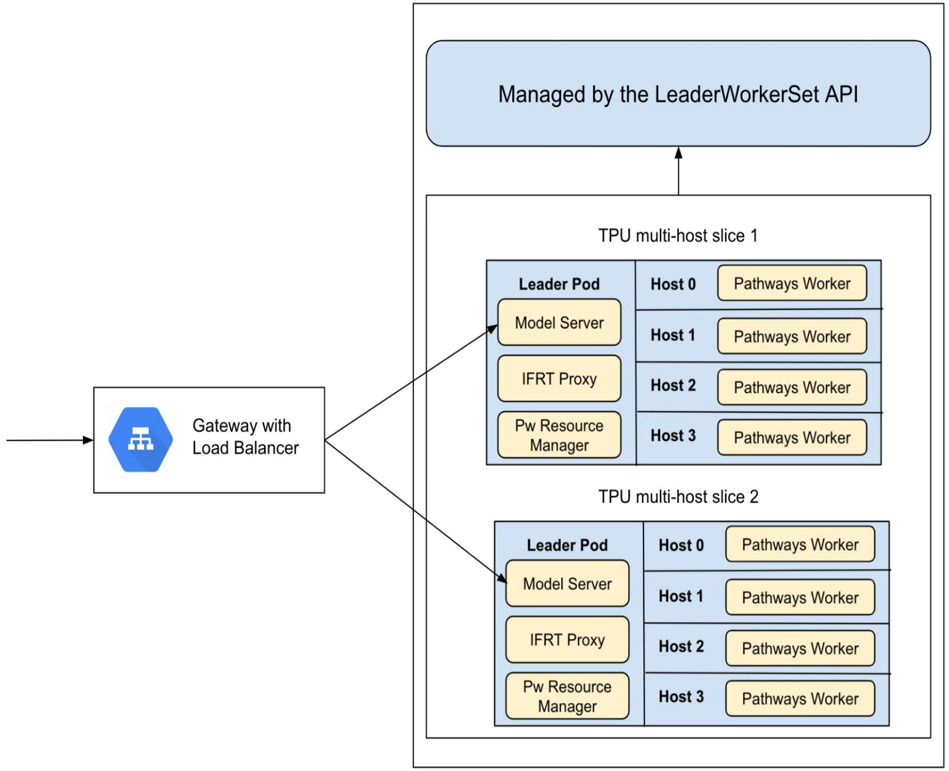
- Two A3 nodes with 8 H100 GPU chips using compact placement (co-locating to minimize cross node communication latency)
- vLLM + Ray as the ML runtime
- Llama3-405B model with tensor parallelism = 8 (# GPUs per node) and pipeline parallelism = 2 (# of nodes)
- all_gather happens within the node over nvLink; cross node data transfer over datacenter network links (or RDMA if available)



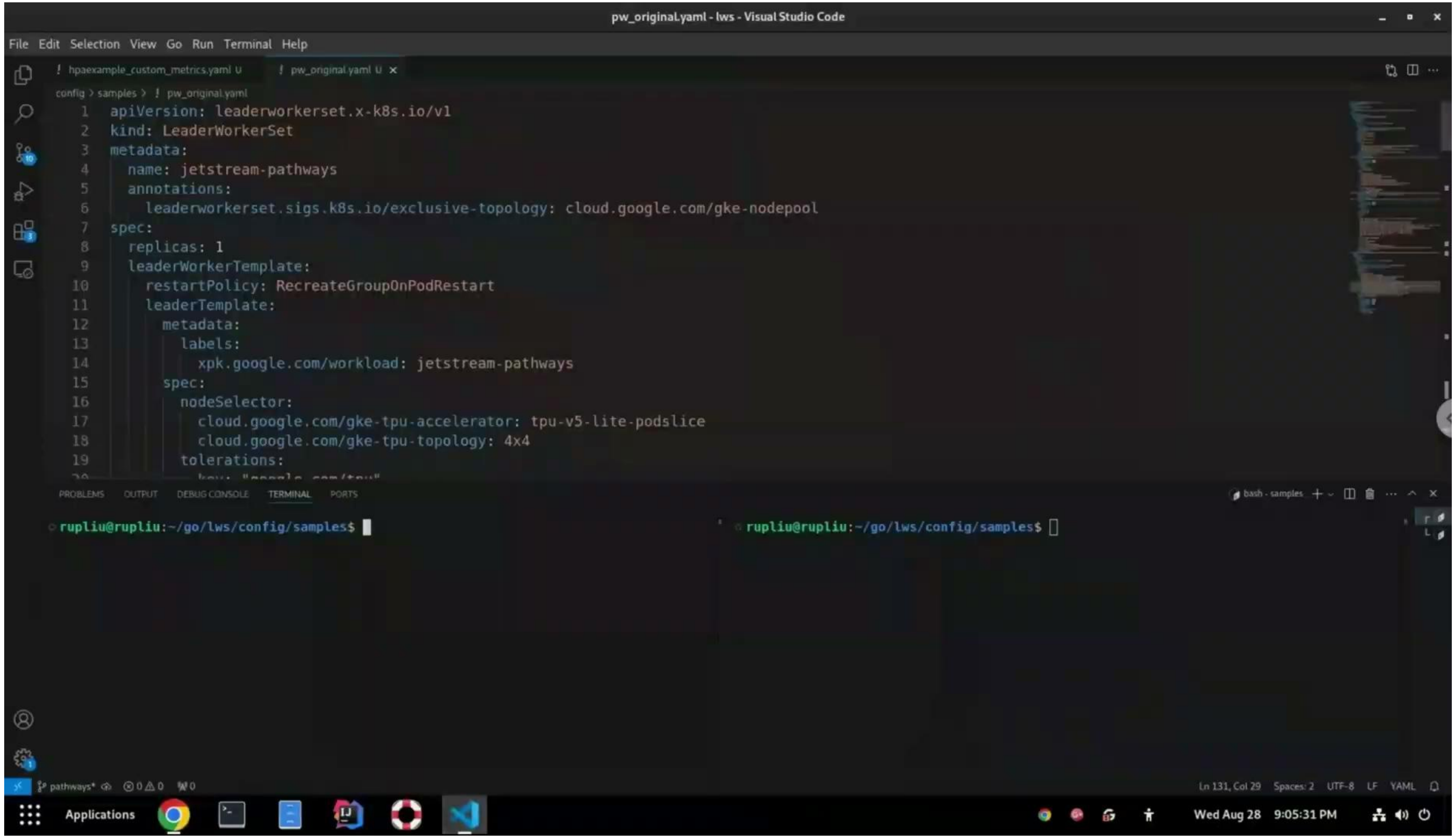
Multi-node inference on TPU

Infrastructure & runtime setup:

- One TPU slice of v5e-16 (5 pods per replica running on 4 node)
- JetStream (TPU inference server) + Pathways (distributed compute substrate)
- Llama3-405B model with tensor parallelism
- Cross hosts communication through InterChip interconnect (ICI)



Multi-node inference on TPU



pw_original.yaml - lws - Visual Studio Code

config > samples > pw_original.yaml

```yaml
apiVersion: leaderworkerset.x-k8s.io/v1
kind: LeaderWorkerSet
metadata:
  name: jetstream-pathways
  annotations:
    leaderworkerset.sigs.k8s.io/exclusive-topology: cloud.google.com/gke-nodepool
spec:
  replicas: 1
  leaderWorkerTemplate:
    restartPolicy: RecreateGroupOnPodRestart
    leaderTemplate:
      metadata:
        labels:
          xpk.google.com/workload: jetstream-pathways
      spec:
        nodeSelector:
          cloud.google.com/gke-tpu-accelerator: tpu-v5-lite-podslice
          cloud.google.com/gke-tpu-topology: 4x4
        tolerations:
          key: "google.com/tpu"
```

```
rupliu@rupliu:~/go/lws/config/samples$
```

```
rupliu@rupliu:~/go/lws/config/samples$
```

File Edit Selection View Go Run Terminal Help
hpaexample_custom_metrics.yaml U
pw_original.yaml U
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
bash - samples
pathways*
Ln 131, Col 29 Spaces: 2 UTF-8 LF YAML
Applications
Wed Aug 28 9:05:31 PM

Disaggregated Serving

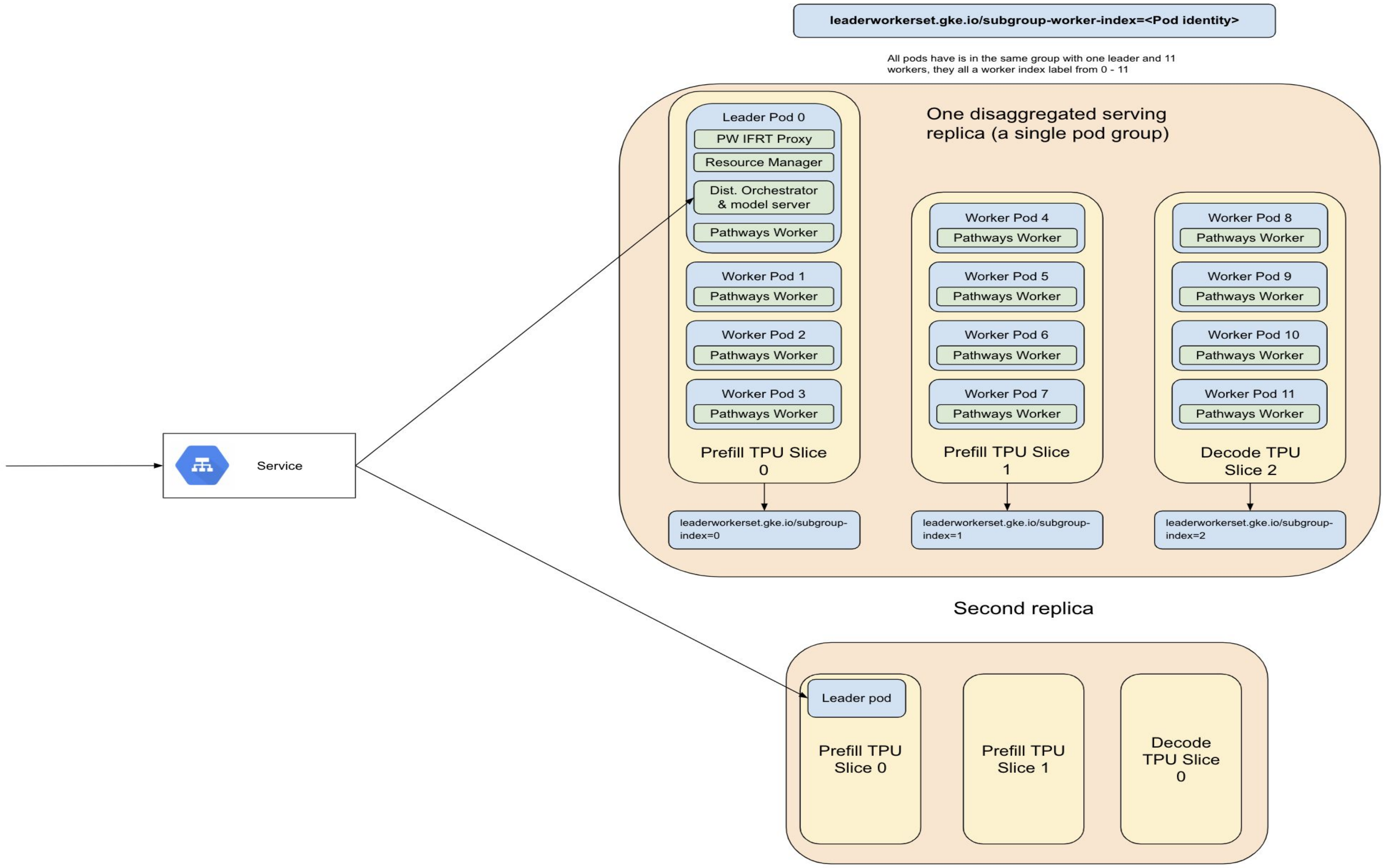
The processing of a LLM inference can be split into two phases: **Prefill & Decode**.

Prefill is more **compute bound** while decode is more **memory bound**. The two phases will interfere with each other running on one server which causes resource inefficiency and unpredictable latency.

To address this problem, disaggregated serving splits prefill & decode stages to run separate servers. This brings the following benefits:

- Reduce prefill/decode interference. Prefill usually work at batch size 1 and decode phase has much larger batch size, whenever there is a request coming in, the decode loop will be interrupted to prefill the new request
- Reduce time to first token latency.
- Enable heterogeneous VM types for prefill/decode that better matches the compute characteristics of each phase
- Better overall resource efficiency and lower cost

LeaderWorkerSet helps orchestrate this type of multi-node serving paradigm


leaderworkerset.gke.io/subgroup-worker-index=<Pod identity>
All pods have is in the same group with one leader and 11 workers, they all a worker index label from 0 - 11
One disaggregated serving replica (a single pod group)
Service
Leader Pod 0
PW IFRT Proxy
Resource Manager
Dist. Orchestrator & model server
Pathways Worker
Worker Pod 1
Pathways Worker
Worker Pod 2
Pathways Worker
Worker Pod 3
Pathways Worker
Prefill TPU Slice 0
Worker Pod 4
Pathways Worker
Worker Pod 5
Pathways Worker
Worker Pod 6
Pathways Worker
Worker Pod 7
Pathways Worker
Prefill TPU Slice 1
Worker Pod 8
Pathways Worker
Worker Pod 9
Pathways Worker
Worker Pod 10
Pathways Worker
Worker Pod 11
Pathways Worker
Decode TPU Slice 2
leaderworkerset.gke.io/subgroup-index=0
leaderworkerset.gke.io/subgroup-index=1
leaderworkerset.gke.io/subgroup-index=2
Second replica
Leader pod
Prefill TPU Slice 0
Prefill TPU Slice 1
Decode TPU Slice 0

Getting Involved

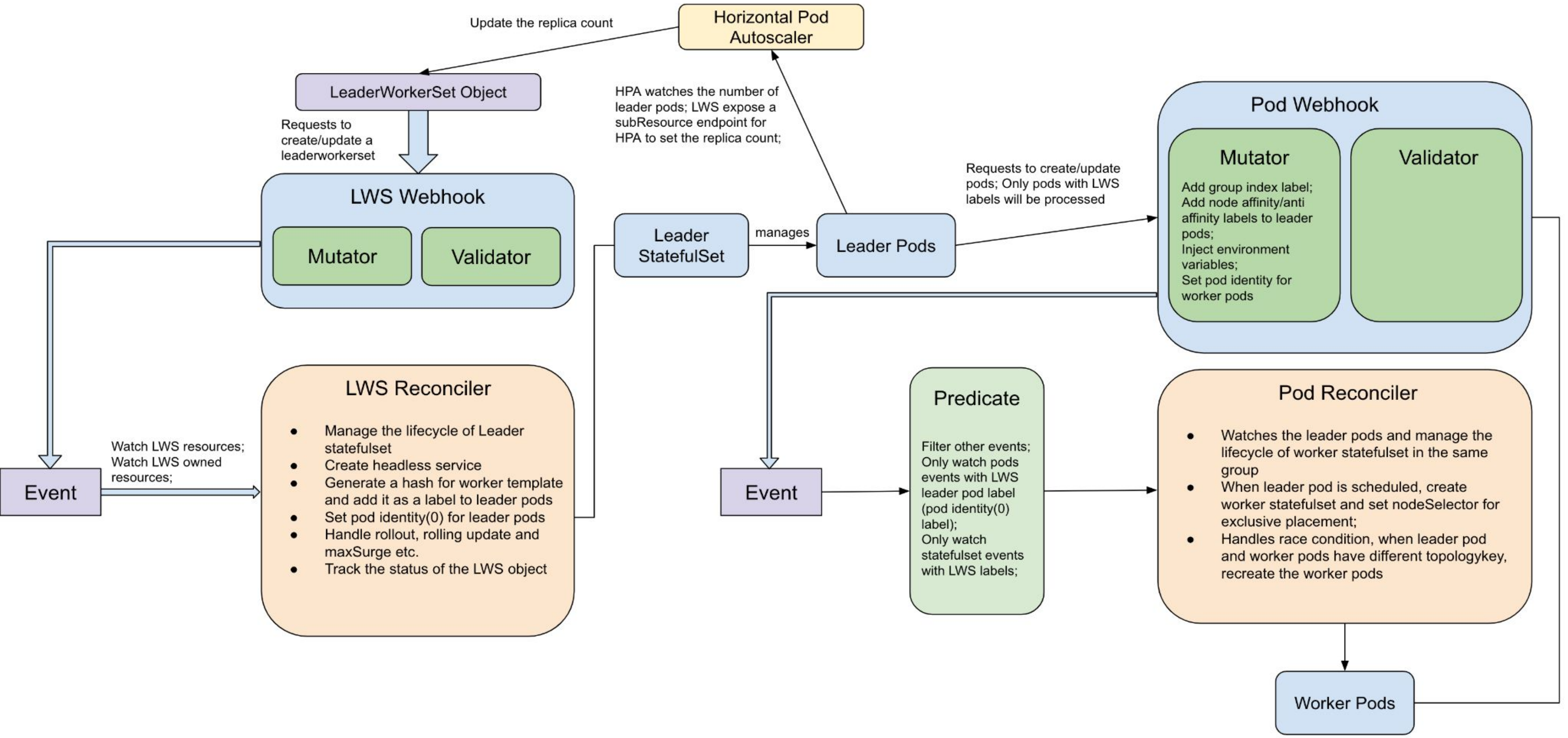
LeaderWorkerSet

- https://github.com/kubernetes-sigs/lws
- A Serving WG sponsored project

Serving WG:

- Biweekly meetings, Wednesdays at 9:00 PT (Pacific Time)
- slack.k8s.io #wg-serving
- wg-serving@k8s.io
- git.k8s.io/community/wg-serving

Implementation details


Horizontal Pod Autoscaler
Update the replica count
LeaderWorkerSet Object
Requests to create/update a leaderworkerset
HPA watches the number of leader pods; LWS expose a subResource endpoint for HPA to set the replica count;
LWS Webhook
Mutator
Validator
Leader StatefulSet
manages
Leader Pods
Requests to create/update pods; Only pods with LWS labels will be processed
Pod Webhook
Mutator
Add group index label; Add node affinity/anti affinity labels to leader pods; Inject environment variables; Set pod identity for worker pods
Validator
Event
Watch LWS resources; Watch LWS owned resources;
LWS Reconciler
Manage the lifecycle of Leader statefulset
Create headless service
Generate a hash for worker template and add it as a label to leader pods
Set pod identity(0) for leader pods
Handle rollout, rolling update and maxSurge etc.
Track the status of the LWS object
Event
Predicate
Filter other events; Only watch pods events with LWS leader pod label (pod identity(0) label); Only watch statefulset events with LWS labels;
Pod Reconciler
Watches the leader pods and manage the lifecycle of worker statefulset in the same group
When leader pod is scheduled, create worker statefulset and set nodeSelector for exclusive placement;
Handles race condition, when leader pod and worker pods have different topologykey, recreate the worker pods
Worker Pods