`Can we get more granular?`**Zoom Link: https://zoom.us/j/9955436256?pwd=Z2FQWU1jeDZkVC9RRTN4TlZyZTBHZz09**

Possible ideas/future discussion:

- Leverage hibernated nodes as a mechanism for warm-pooling (https://github.com/kubernetes/autoscaler/pull/6202).

Next topics?

- Is there any health/unhealthiness signal coming from the server the autoscaler or LB should be aware of? (One example was the amount of work and their states in the queue)
- As further optimizations land (eg. disaggregated serving) how do some of our existing assumptions change?
    - One example highlighted by Evan in Clayton's doc is TGI will be making partial progress on queue items which could mean the "traditional" queue depth metric isn't what we think it is.
- What are key differences between "rank-and-file" deployments vs sophisticated high-traffic large deployments?
- How do we autoscale and load balance services backed by diverse backends (meaning some backends are slower than others since they may use different accelerator types), queue length thresholds for example is not going to be the same, and likely needs to be set per backend "type"
- Is there opportunity to observe GPU metrics (eg. DCGM) to derive a "busyness" signal? Or even to help validate that autoscaling is doing its job?
    - eg. Power Draw? (DCGM_FI_DEV_POWER_USAGE)

1 Aug 2024 | WG-Serving: Autoscaling Workstream

Attendees: calendar@kubernetes.io wg-serving
Attached files: WG-Serving: Autoscaling Workstream
Recording: https://www.youtube.com/watch?v=jqRPZTfoXSY

Notes

- All about Model Caching and startup times
    - [Jeff] Container images are big
        - Sidecar - High parallel reads from buckets
        - [kunjan] leverage container image caching for model weights.
        - Dragonfly & Nydus for P2P and Lazy load [these technologies exist but are not designed for 100GB+ data and are operationally challenging]
        - A problem with Docker having model weights is it is single-threaded (gzipped).
    - [Clayton]
        - Easy to use GCS, S3, download directly from URL.
        - Platform team may identify that this is just an artifact distribution system.
            - Some prefer using container images.
            - ML Engineers may not want or need to care about images.
        - There may be “disadvantages” about using images, so until we mitigate these, it’s hard to nudge folks in this direction.
    - [Jeff]
        - OCI spec doesn’t allow parallel uploads of layers (Clayton: but the Docker v2 distribution spec DOES)
    - [Eduardo]
        - OCI Artifacts protocol
        - Should we standardize pulling these from Hugging Face using OCI?
    - [Clayton]
        - There is an appetite to improve OCI images. Should this group be pushing this?
    - [Kunjan]
        - Tried both approaches.
        - Like standardizing pulling from Hugging Face.
        - Initial problem: building weights into an image.
        - Build pipeline to package weights into image was long due to long push from local to repository because the large image.
    - [Jiaxin]
        - Easy to re-use tools to fetch or push artifacts.
        - Splitting the file into multiple chunks is likely to be helpful
        - Any benchmarking using GCS or S3 using multipart?
        - [Clayton] pull and push can use multipart
            - harder is generating the hash
            - how many pulls over the next few years will be pulling base images?
            - can do multiple layers
    - [Jeff]
        - Like the idea of images but none of the existing technologies apply to this use-case (100GB vs 1-2GB).

        - Cloud Container registries are slow.
        - Maybe a cache layer between you and the registry?
        - Pushes are also slow.
        - potentially have custom checkpoints with fine-tuning data
            - could maybe solve this with object store
    - [clayton]
        - common use cases - use base images
        - everything else use object store
        - some do peer-to-peer pulls which is very fast
    - [Jiaxin]
        - From a system design, registry has a lot of overhead.
    - [Eduardo]
        - Let's push this into the core meeting.
    - [Clayton]
        - Haven't gotten around to model caching.
    - [Evan]
        - I like the idea of OCI and registry because of the features, but in my industry you don't have access to commercial cloud registries in many cases.
        - The more you can stay within-cluster the better. Local Minio S3 solutions are most common.
- [ray/kunjan] https://github.com/kubernetes/enhancements/tree/master/keps/sig-node/4639-oci-volume-source#summary
    -

Jul 25, 2024 | WG-Serving: Autoscaling Workstream

Attendees: calendar@kubernetes.io wg-serving
Attached files: WG-Serving: Autoscaling Workstream
Recording: https://www.youtube.com/watch?v=bW5semG5u8c

Notes

- [Ashok] Thoughts on benchmarking
    - [PUBLIC] Benchmarking Workloads for Performance Evaluation and Autos…
    - Should we create a benchmarking operator on Kubernetes?
        - Would enable an easier way to run benchmarking in a continuous fashion, along with autoscaling and load balancing.
        - Should this be a new project for the Serving WG?
        - Q: Normally when you're benchmarking, you wouldn't necessarily want an operator running on a continuous basis?
            - Yeah, usually you run it before you qualify a new model, for example.

        - A continuous case might be like, you want to monitor performance on an ongoing basis. Maybe you change a metric, and you want to see how it's going in real time. This would allow you to generate traffic. It wouldn't be something always running, but it wouldn't be only a once-and-done kind of thing either. It could be like a CI/CD step. For qualifying changes, for example.
    - 2 pieces of Feedback:
        - The ability to generate different traffic to mock up production traffic seems useful. We can probably do more with statistics we provide through it.
        - On the client side, could we provide like service tracing data in the future? Analysis with the benchmark together in one step?
        - [Ashok] The point on metrics is interesting. There are some projects I've seen around adding tracing to workloads like this. I agree it would be good to have. Can look into it.
    - [Clayton] Q: Looking at the benchmarking API, I'd say there are 2 parts of this api bundled together. Successful things usually have a few dependencies, but makes the core problem tractable. For a use case example, if I wanted to contrast a self-hosted model vs a non-self-hosted model, this solution would be clunky for that. I think the coupling in the API as presented, there seems like a missing part. What are the actual requirements for doing a great operational benchmark? And that should be referenced from elsewhere - it's not a Kubernetes thing. The Kubernetes Operator would be a next step. IE:
        - Step 1: Come up with a great operational benchmark
        - Step 2: Integrate with K8s
        - Step 3: ???
        - Are there others in the group interested in this?
        - [Jeff] Having LLM-specific benchmarking built in to a benchmarking tool sounds useful.
        - [Eduardo] I was just having a discussion around this. It would be good if we, as a WG, could invite the ML Commons to talk with us. They're the standard. If we could agree on a benchmarking API, making use of the ML Perf suite of benchmark, that would be great. They're mostly academic folks who are very specialized.
            - [Clayton] "Yes and." I think ML Perf is serving a different customer than the one I care about - which is a platform team running serving workloads. The framing might be, "we have a user - someone running inference at scale. That user has a business objective around selecting a model for quality, achieving a reasonable cost, running stable-ly in production." These users are focused on golden signals. With that user in mind, how can we best serve/enable that user?

        - [Eduardo] Reading ML Perf papers is difficult. The reports are opaque. I get what you're saying, but they built something. If we could build on it, that could be really helpful.
        - [Clayton] We shouldn't re-invent things, we should focus on the benchmark one does after. We care about production use cases.
        - Eduardo to add comments to doc.
        - [Jon Li] ML Perf is for testing single-replica performance in constrained environments. Ashok's proposal is about testing an entire system. Really an end-to-end type of benchmarking, which is fundamentally different from ML Perf.
        - [Clayton] 2 variables ML Perf doesn't consider: How much hardware can I bring? Latency profiles. We're trying to help people test a couple of key variables against objectives over time and across a system. I want whatever hardware I've chosen to meet the required objective - which is kind of opposite of ML Perf's approach.
        - [jiaxin] I think clayton's point is to help end user choose right platform/deployment, given optimization goal. optimization goal related metrics could be (cost-million token/dollar. latency-tpot, throughput - token/s etc) or even multi-dimensional optimization
            - [jeff] Or at least to give the platform team the ability to see how config changes impact these parameters.
    - Next meeting is model caching!
- Possible other topics for discussion:
    - [public] KV-aware LLM Autoscaling and evaluation metrics
    - (see above)
- Next week:
    - As discussed in wg-serving yesterday, we'll dive into workload startup improvements.
    - Maybe brainstorm today?

Action items

- [ ]

Jun 27, 2024 | WG-Serving: Autoscaling Workstream

Attendees: calendar@kubernetes.io wg-serving
Attached files: [PUBLIC] Kubernetes LLM Inference Autoscaling Examples
Recording: https://www.youtube.com/watch?v=hosn5IcuAbY

Notes

- We'll change the timeslot for this based on some feedback from WG-Serving group
  - watch out for poll/interest thread in slack
- Call for proposals?
- Anyone aware of any open issues or PRs that might be relevant to this group?
- [Jiaxin] LLM metrics and evaluation metrics from Bytedance.
  - Preparing a doc about metrics they are using for autoscaling, how they are transforming latency metrics into something consumable by autoscaler. Will share by next week.
  - Whitebox - checking kv cache
  - Two steps:
    - utilization
    - prediction (overall KV demand)
  - GPU metrics are not linear.
  - Believe their proposal is a generic way to solve this regardless of hardware.
  - How to efficiently evaluate HPA settings are working correctly?
    - introduced SLOs for resource utilization, pod count changes and overall resource changes
  - [Ashok]
    - Have you tried autoscale on queue size? And how did KV-Cache or GPU utilization look?
      - [Jiaxin] Depends on the engine used. Page attention to improve throughput which will cause batch size to be very large. This makes the queue quite useless as there won't be very many pending requests.
      - [Jiaxin] If there are pending requests, it means cache usage is quite high. Latency is likely already very high.
      - [Jiaxin] By the time queue size is high, latency will already be high.
      - [Jiaxin] For static batching, queue size will be more interesting.
    - Which GPU utilization metrics did you find were most useful?
      - Found some were not great to autoscale.
      - [Jiaxin] Not a good metric. Even with 1 SM Active, GPU utilization will still be 1.
      - [Jiaxin] SM Occupancy.
      - These metrics are not linear and are not indicative of how busy the server is.
      - These engines are still not using the GPU's concurrency fully.
      - Gave up on using these metrics and moved to the whitebox.

    - [Abdullah]
        - Is it with HPA?
            - [Jiaxin] Yes, still using HPA.
            - Provisioning efficiency.
            - Reading the metric to be used for autoscaling - prometheus -> adapter -> autoscaler. This path is pretty slow.
            - Any reactive way will not fix this issue (which is how autoscaling work).
            - Predictive time series which works well for traditional workloads. Working on applying this to Gen AI but is tricky due to non-linear relationship.
            - Want to shortcut Prometheus stack by reading metric from the pod directly. Work inspired by fast system.
        - Are there concrete proposals on how we can improve HPA?
            - Problems will be mentioned in the doc.
            - Metric scrape intervals can be adjusted but not sure if this is the best solution due to resource and load consumption.
- [Ashok] How do you evaluate performance today? What benchmarking do you use? There are ways we can look at standardizing it on Kubernetes.
    - Each model server seems to do their own benchmarking.
    - For Bytedance, what do they do?
        - mock data for internal workloads
        - public benchmarks - share gpt
        - Evaluation - have some tools to quantify results, oscillations, resource usage, etc… Produces scores for the performance.
        - Test runs for a few hours.
        - Haven't made it fully automated.
        - Using Locust.
        - Trying to find more tools that mock traffic patterns, Locust is simple as it just increases concurrency to the target level.
            - Would like to see peak and off-peak.
        - [Abdullah] TGI Benchmarking tool
            - [Ashok] Using Locust but also running into challenges around QPS and traffic patterns. TGI one was more specifically focused on TGI, not easy or possible to use other model servers. Script that comes with vLLM is useful. Can we do better in this space? Can the community help build something? Have some thoughts on this and will try to put together a proposal.
            - [Evan] Would be a concrete first step.
        - [Evan] Any more details about evaluation?
            - Initial 3-4 metrics, they will share.
            - Composite metrics to define whether a server is free or not. Planning on writing papers on some of these more advanced solutions.

        - How to calculate oscillations, etc.. from production. Will share next week.
    - Are there any improvements that we can make?
- [Jiaxin] HPA Autoconfiguration?
    - HPA v2 supports a lot like multiple metrics.
    - For platform owners, it is hard to configure these optimally.
    - Space-search problem. So many parameters to be tuned.
    - Any work that can be done to auto-tune those metrics to the user-defined goal? This could possibly make parameter generation more straightforward.
    - [Evan] Auto-tuning work would dovetail nicely with the aforementioned benchmarking tool; if users can simulate their expected traffic, you can use this to drive the tuning.

Action items

- ☐
- ☐
- ☐

Jun 20, 2024 | WG-Serving: Autoscaling Workstream

Recording: https://www.youtube.com/watch?v=RdoNHNNfDfM

Attendees: calendar@kubernetes.io
wg-serving

Notes

- How do we define the critical inflection points where the model server changes behavior when some characteristic of the incoming traffic changes. And how should autoscaling care or try to prevent this from happening?
    - [Clayton] First transition point where you hit a roofline. Most poeple don't operate at the optimal point but instead beyond that.
    - [Robert] Maximizing the accelerator is not possible in an LLM workload due to memory-boundedness with regards to decode phase. The piece of the accelerator that is being maxed out is the memory bandwidth.
        - There is no way to "overuse" the accelerator in an LLM workload.
        - You are bound by the time you are moving the model weights across memory.
        - You can measure some utilization signals (like memory bandwidth utilization). Aspirational goal is to make MBU as high as possible.
        - "I want to make MBU as high as possible for a particular latency objective".

        - [Clayton] Variables people have available are {model, model server}, next variable is how much hardware available to expand horizontally. It is the model server's job to take the traffic it is given and extract the most performance. Level above that is "how as a Kubernetes project, what are the common problems that we can handle". What signal can we give to the model server? Queue depth and admission.
        - [Clayton] Model servers can leverage the queue.
        - [Robert] Under the critical point, increasing load doesn't hurt the TPOT too much. To maximize efficiency, we want to add load right up to that critical point and then expand horizontally.
            - Likes the queue time -> server is either keeping up with queue OR falling behind.
        - [Clayton] One of the pains with HPA is you need to experiment to find the constants.
            - More painful in LLMs because there is a large startup cost.
            - Admission strategy is important.
            - Are we leveraging compressibility? Statelessness of work?
            - There are advantages to hedging requests (canceling as needed).
            - Autoscaling is one layer, how many replicas.
            - Higher level is how to balance traffic (and cancel traffic).
        - [Robert] CharacterAI Blog (https://research.character.ai/optimizing-inference/)
            - Prefix caching scheme and as a result they need to route requests to specific servers.
            - In the Gateway, having traffic stickiness would be useful.
        - [Clayton] Belief that there will be more state in model servers in the future.
            - eg. eventually will have some kind of "session"
        - [Clayton] Deployments work in the assumption that everything is fungible. StatefulSets do not.
            - Autoscaling down individual instances? If session affinity is more important might need to think about which instances need to be scaled down. Pod Deletion Cost, etc..
        - [Robert] How to quick scale up? These inflection points are "once you reach equilibrium".
            - For autoscaling, there is usually a gradual change from lower request rate to higher request rate.
            - When moving from RequestRate1 to RequestRate5 over 30s, it will take roughly 1 minute for the server to reach equilibrium.
            - TTFT is more of a trailing signal. Queue is more forward looking.
            - When you are transitioning from healthy to unhealthy state, the queue will be a good signal.
- [Justin] What is the expected of this workstream/workgroup?
    - Understanding and getting people aligned on serving to make changes to Kube.
    - [Justin] Should we aim for a blog post, benchmark or reference implementation?

        - [Clayton] Everyone is running their own benchmark. Often times benchmark runs are not specifying latency.

Action items

- [ ]

Jun 13, 2024 | WG-Serving: Autoscaling Workstream

Attendees: calendar@kubernetes.io wg-serving
Attached files: [PUBLIC] Kubernetes LLM Inference Autoscaling Examples
Recording: https://www.youtube.com/watch?v=jxVoa4SxDYo

Notes

- [Clayton] Break the doc into 2:
    - Heterogeneous hardware
        - Maybe because of capacity issues or cost, you need to run models on a variety of hardware.
        - You can route requests to different requests to different accelerators depending on scenario (eg. small requests to older accelerators).
    - If you have homogenous hardware and single goal (A1 and A2 in the doc).
        - Latency objective in all use-cases - “What is the best I can get as cheaply as possible”
        - Novel thing with LLMs, bigger input or output, the longer it takes.
        - More parallel requests causes slowdown for everyone.
        - Alternative is queuing to not overload concurrency.
        - Almost all LLM model servers have queuing constructs.
        - Admission decision (done before adding work to the queue).
        - Depending on your objective, your admission and autoscaling metric will change (as described in the doc).
- [Rob]
    - What is different for LLM from typical inference:
        - continuous batching
        - internal management of queuing and concurrency creates some surprising dynamics
        - LLMs are typically memory-bound, as you increase concurrency you generally don’t see a big effect on latency.

        - But you'll hit these critical points where too much time is spent processing the prompt.
        - All of these nuances makes it difficult to set up the autoscaling correctly.
- [Clayton]
    - One assumption we had last year was continuous batching is the state of the art.
    - vLLM has open PRs for other capabilities (chunked prefill, batching, etc…)
        - [Rob]
            - Chunked prefill (mixed-wise batching) will be turned on by default, will smoothen these critical points. In beta stage now. Will make the thresholds higher. Also known as dynamic splitfuse. Has other names as well in industry. SARATHI-serve.
            - Splitwise - prefill & decode disaggregation / disaggregated serving.
            - Some of these require the KV-Cache to move between accelerators which is not something that is solved.
                - Discussed in multi-host inference.
                - Having good network topology for this reason is one of the requirements in multi-host.
        - [Clayton]
            - What is the timeframe for these big changes to land?
        - [Rob]
            - These optimizations are aimed at very high-request rate setups.
            - Rank-and-file LLM deployments are likely to not adopt these (as they are more complex).
        - [Clayton]
            - There's a strong advantage to densifying your model servers.
            - Lots of Lora on top of model servers.
            - We might want to categorize the various uses (very high scale vs enterprise).
        - [Jon]
            - Moving KV caches between accelerators, using the ICI for data center network movement?
                - Let's discuss this in multi-host
        - [Clayton]
            - What are the core things that are durable over the next year as techniques change inside the model server?
        - [Rob]
            - Moving from a request rate of 2 to 3 can have surprisingly big differences (example of critical thresholds that can have a big effect). Getting server in an equilibrium is key.
            - Concurrency in the system is the wrong metric to look at (it's only a proxy of what you actually care about) - are you moving from a healthy state to an unhealthy state?
            - Can we get more granular?

        - vLLM states: running, waiting, swapped (on pause)
        - You would want to autoscale on waiting and swapped queue.
    - [Clayton]
        - Next step:
            - Criticality thresholds
                - Define the various thresholds, and the impact of transitioning across them (i.e. to rooflines for compute/memory, to memory exhaustion in kvcache, to unhealthy metrics)
                - Filling up prefill (choking the server outage) / prefill saturation
            - Healthy/unhealthy simplified perspectives coming from model servers
            - Assessing load (how do you at a high level direct traffic to model server instances?)
                - eg. Requests going through the same Lora should be focused to the same servers.
            - What's the idealized model for the queue on a model server, and what responsibilities for queue above model servers can solve things the model servers can't
                - I.e. how do abstract
            - Which workstream will cover traffic / queue management across multiple servers? Can be autoscaling to start
            - Take into account assumptions from Rob into existing docs, set the current doc as a "today's state" and draft a new doc with folks on "next state"
    - [Evan]
        - Does vLLM currently expose these metrics?
            - [Rob] Yes, we expose these metrics. Rob is the one who implemented Prometheus into vLLM so can help with adding additional metrics.
            - [Clayton] Queue time is important for request hedging (send the same request to two different model servers).
        - [Rob] What is happening during these critical points? (eg. GPU becoming compute bound?)
            - LLM server is a background service running in a loop.
                - Processing new work
                - Processing decode, generate 1 token in each request.
            - Adding the 50th item to the batch has no effect on inter-token latency.
            - You can add more requests to decode, this is very scalable.
            - Prefill times are interrupting decode batches.

    - If request rate is too high, prefill starts to choke out the decode.
    - TPOT will almost never go up because servers respond to this by growing a queue because it keeps itself safe by only allowing a certain number of requests through at a time.
    - For some use-cases, TTFT is more important.

Action items

- [ ]

May 30, 2024 |
WG-Serving: Autoscaling Deep Dive on Warm Replicas Foll…

Attendees: Ray Wainman wg-serving swghosh@redhat.com hebaelayoty@gmail.com mwaykole@redhat.com eduardoa@nvidia.com harpatil@redhat.com guyjtempleton@googlemail.com Kevin Klues sonju0427@gmail.com filippespolti@gmail.com smodeel@redhat.com pfobos@gmail.com Sourabh Deshmukh Abdullah Gharaibeh andrey.velichkevich@gmail.com Ming Zhu Sergey Kanzhelev skontopo@redhat.com s.kanzhelev@gmail.com Clayton Coleman fkrepins@redhat.com mariusjoffre@gmail.com bindaasravi@gmail.com seedjeffwan@gmail.com maszulik@redhat.com somalley@redhat.com smodeel@redhat.com Yuan Tang ~~Daniel Klobuszewski~~

Attached files: WG-Serving: Autoscaling Workstream

Recording: https://youtu.be/XU9BCqQZYo0?si=goF6Ti7W_LlGT3mf

Notes

- 2 major problems: (Jiaxin)
    - Scaling accuracy: whether the autoscaler we can be accurate to the current load
        - More important
        - HPA is in the feedback loop
        - Key issue is on the metric side, DCGM metrics, instead looking into the inference worker
    - Provisioning efficiency: how fast we can bring everything up to serve traffic
        - Model loading
        - Warm replicas would sit here
            - Some providers allow you to use warm pools
        - 2 parts:

        - Infrastructure side - how fast can a node brought up, pod scheduled
            - Sidecar
                - Most inference engines can load the model from remote sources. Platform team can optimize this.
        - Application side - how fast can the replica come up
            - Some optimizations include loading large amounts of data over PCI-e.
            - More on the application side, on the engine side.
            - Opportunity to do this here?
- Provisioning request:
    - We know about incoming workloads
- NodeClaim (subin, Heba Elayoty)
    - You want to claim nodes based on some metric (cost is predominant)
    - Looks at current inventory - spot instances, instances, etc.. and redeploys workloads to optimize for cost.
    - List of requirements that a customer can customize - toleration, gpu, memory utilization, labels
    - Karpenter will wait for pending pod.
    - No current concept of anticipating work.
- Next:
    - Think about how to reconcile the concept behind ProvisioningRequests across both CA and Karpenter (or at least have similar functionality across both)
    - Continue exploring more signals in the <Pod, Proactive> quadrant from the table below.
    - Think about other pain points in terms of scaling accuracy - right now using DCGM metrics for example is difficult.

Action items

- [ ]

May 16, 2024 |

WG-Serving: Autoscaling Deep Dive on Warm Replicas

Attendees: Clayton Coleman wg-serving pfobos@gmail.com harpatil@redhat.com ahg@google.com Heba El Ayoty Yuan Tang pehunt@redhat.com kaihsun@anyscale.com dtuname@gmail.com smodeel@redhat.com rwainman@google.com mariusjoffre@gmail.com rupliu@google.com areber@redhat.com eduardoa@nvidia.com achandrasekar@google.com vrgf2003@gmail.com

sonju0427@gmail.com supertetelman@gmail.com Rethish Kumar P.S
arangogutierrez@gmail.com ravsharm@redhat.com fkrepins@redhat.com
s.kanzhelev@gmail.com seedjeffwan@gmail.com guyjtempleton@googlemail.com
maszulik@redhat.com ~~aramiss@ziprecruiter.com~~

Recording: https://youtu.be/yi_hA-2OtJI

Agenda:

- Identify ways of amortizing startup time for accelerated inference workloads
  - Keeping a pool of nodes “warm” for scale-up (amortize slow nodes)
  - Problem definition
  - Overlap with other use cases
- Review of other startup mitigation techniques and any open keps or proposals in kubernetes

Notes:

- How to improve workload startup
- Pooling nodes
  - A node that is started but not running the workload is “warm”
- Pooling pods
  - A pod that is started is “hot”
  - If we can amortize
- Using the node object with node feature discovery (NodeFeature CRD) labels to create a desired intent to have a node
  - https://kubernetes.slack.com/archives/C01A82FKSQK/p1712153874968349
  - Declarative signal to that a specific node is needed
    - Need a signal to keep that node around
- Current signal is creating a pod with a set of spec to create a node (“scale up”)
  - Absence of a pod consuming some resources (pod deletion etc) is viewed as a signal to autoscaler to spin down nodes
  - Repack signal can be inferred from the controller type
  - Scale up node signal today is being worked on to align
    - Karpenter signal for node is NodeClaim
    - ProvisioningRequest is CA
- Reactive in HPA and CA and Karpenter
  - Sergey: in batch, if you are thinking about starting the next pods while still running that’s proactive - might also have that use case
  - Sergey: is preemption in scope
- Guy - ongoing discussions in sig-autoscaling to align, Abdullah not aware of many others
  - Clayton: seems we have scope and remit to explore this and help bring requirements / suggestions to the various gruops
- Eduardo: kueue is using pause or onpause, can we expand that?
  - Kueue could be actively managing the “warm” state since it has an idea of the over time

- Abdullah: reservations was an old idea https://bit.ly/k8s-reservations
    - Along the lines of ProvisioningRequest
    - Trigger provisioning as well as reserve
    - Like taints but for part of the node as well
- Clayton: can we think of the state of declarative workload controllers + HPA + pods as a sort of policy (if we know pods take X time to start, can we build aggregate intent)
- Eduardo: we should define the set of signals and how the systems can prewarm
    - What is “ready”
- Ray: We’re brainstorming - to organize this a bit more to generate more ideas to feed next discussion.

Signal Categorization

bold = New signal ideas (or existing ideas not yet implemented)

(E) = Explicit strong signal or action that leads to an expected outcome with respect to autoscaling (eg. setting an HPA CPU% value).
(I) = Implicit signal is something that can/could influence autoscaling but is a result of other explicit actions (eg. setting a Pod Disruption Budget, we say it is implicit because even though a pod could be evicted if it has enough pod disruption budget, it is not a direct reason to scale it down).

Scale Down:

	Reactive	Proactive
Pods	(E) HPA Metric (E) HPA min_replicas (I) Pod Disruption Budget	**(I) Predictive Signal Based on historical usage data?**
Nodes	(I) Under-utilized nodes (E) CA min_nodes	

Scale Up:

	Reactive	Proactive
Pods	(E) HPA Metric (E) HPA max_replicas (E) Keue pause/unpause?	**(I) Predictive Signal Based on historical usage data?** **(I) Workload startup time?** **(E) new warm_replicas parameter on HPA?** **(I) Pod Status? - some state before ready?** **<?gap here?>**

Nodes	(I) Unschedulable Pending Pods (E) max_nodes (CA) (E) limits (Karpenter) (E) NodeClaim (Karpenter)	**(E) k8s-reservations** **(E) ProvisioningRequest (CA)** **<gap in Karpenter?>**

-

May 3, 2024 | Working session for WG-Serving: Autoscaling

Attendees: Clayton Coleman, wg-serving, Sachin MV, hebaelayoty@gmail.com, Kevin Klues, farceo@redhat.com, guyjtempleton@googlemail.com, Abdullah Gharaibeh, Davanum Srinivas, pehunt@redhat.com, sonju0427@gmail.com, arangogutierrez@gmail.com, mariusjoffre@gmail.com, denizkavuk86@gmail.com, Ray Wainman, Sergey Kanzhelev, Tran, Nick, Jonathan Innis

Notes
Recording: Working session for WG-Serving: Autoscaling (2024-05-03 12:34 GMT-4) (shared with wg-serving)

- Discuss novel user scenarios that are different for inference or accelerators than regular workloads
    - What is in and out of scope (concrete examples)
    - What is the key user scenarios
        - [Clayton] Latency related use cases for large models - scale to a target latency objective
        - [Eduardo] Takes more time to ask the ingress to add an IP to the frontend than for the workload to come up - autoscaling is faster than the infrastructure provisioning - pod startup can be very fast, but the infrastructure programming is actually slower (i.e. model as a service)
        - [Sergey] Ray mentioned scaling down specific pods, because they're orchestrating their own pods on top of Kubernetes
            - In the same vein: Node evacuation to release the accelerator on specific node
        - [Dims/Nick/Jonathan] What can cluster autoscaling do?
            - Getting GPUs and other accelerators faster, tearing things down faster
            - Ollama use cases - doesn't support multiple models at the same time, just merged code for parallel processing - challenges that they are seeing, not just enterprises

            - Clayton: suggesting warm pooling as ways of mitigating cluster autoscaling - but we also want the better preactive and prewarming
        - [Jonathan] Relevant to this conversation: https://github.com/kubernetes/autoscaler/pull/5848
            - [Clayton] Gang scheduling will be important for multi-host inference
        - [Clayton] Is latency-based objective autoscaling something other people hear a lot?
            - What does the workload need to communicate to the infrastructure?
                - E.g. ollama - each model are different and needs different things - drivers, replicas
                    - Will need a mix of models for different applications
                - Lots of workloads living together
        - [Mathis] GPU Memory capacity autoscaling (packing multiple models into accelerators dynamically) is important
        - [Dims] cost is important
            - [Eduardo] big and small models
                - GPU slicing vs GPU multiple use
                - DRA for multi-node GPU management
                - Clayton: example of GPU sharing is important - over time users will expect
            - Eduardo: have discussions with framework developers - how do we expose APIs to make it easier to define what a job needs - what resources
                - End users don't always know what to expose - framework developers know how to use it
            - Dims difference between the roles (people know about models, people who fit it into kubernetes)
            - Clayton: create a matrix of different types of users and use cases
- Identify key problems and what everyone's individual priorities / focus are
    - Identify insurmountable problems vs those with workarounds
    - [Eduardo] mention about pods as a group
    - Action: folks identify what groups of users and use cases they are focusing on
- Review work in progress in Kubernetes today and any 1.31 items to accelerate
    - [Eduardo] Warm pooling and pause pods
    - Clayton: suggestion metrics across use cases
        - Which adapters are well supported
    - Performance characteristics / reacting quickly enough
    - Scale to zero for cpu workloads (are there interested parties on the GPU
- Action item for the first couple months is to collect use cases, to be able to find common problems and key solutions to address as much as possible.
    - Examples of framework users to solicit

        - Notebooks on Kubernetes
        - Model servers and best practices
        - Ray and other orchestrators
        - Sophisticated users building on Kube
    - Clayton: find solutions that multiple use cases - if we can divide and conquer but then pair to push solutions over the line

WG-Serving: Autoscaling Workstream

Note: members of dev@kubernetes.io have commenter privileges and members of wg-serving@kubernetes.io have edit access.

Key problems to address

- Startup time is slow
    - Infrastructure startup (reaction times of autoscaling, node startup)
        - Slow node scale up on managed providers
        - Autoscaling reaction times
        - Very fast pod startup
            - Kubelet faster probing time for readiness
            - Kubelet status propagation for the first ready event to apiserver is slower than it should be when nodes are busy
    - Workload startup (pod scheduling, disk mount, image/model pulling, model load)
        - Downloading images (1 min to 10 minutes)
        - Downloading models on startup (1 minute to 30 minutes)
            - ReadManyOnly PVC
            - In-cluster fast storage
            - Bundling models with VM images (cloud provider specific)
            - Loading models from container images to disk
                - Kserve model car (Feature Track)
                - Container image volume mount type (see https://github.com/kubernetes/kubernetes/issues/831, open since 2014)
                - [Mrunal] Recent KEP https://github.com/kubernetes/enhancements/pull/4642
- Friction
    - Using custom metrics is hard
    - Inconsistency of metrics across model servers
- Being able to autoscale up and handle whatever hardware we get
    - Asking for 1xA100 or 2xL4 on the same pod spec
    - Handling different operational characteristics of node types when bursting into spot use cases (drain time for spot instances different than masters)
    - Being able to scale down specific pods on hardware you don't want anymore

- Challenges with auto-scaling metrics
    - System metrics non linear
        - Metrics like DCGM_FI_PROF_SM_ACTIVE cannot spatially represent the busyness of the GPU.
    - Different model servers are somewhat inconsistent in which metrics they expose
        - I.e. queue length / computation can approximate load, but depend on variable admission control for requests
        - Can we suggest standardized metrics for model servers that would make autoscaling in kube easier and work with model server projects to adopt?
    - Custom metrics adapters for Kube are all over the place
        - Are there gaps in pod metrics that could reduce friction in use across distros? I.e. tenancy concerns for adapters
    - No native support for non-gauge metrics (eg. if you wanted to use a histogram metric like latency directly as a target for autoscaling)
- Inference Engine co-design
    - Inference engines implement various strategies such as KV offloading and batching, which complicate generic autoscaling. Effective integration necessitates co-design to avoid inefficiencies.
- Scale to and from zero
    - Scale to and from zero in Kube is a very old issue https://github.com/kubernetes/kubernetes/issues/484 - can we find common ground on inference for serving to make it work well?
    - Work queue driven autoscaling is easiest because it is resilient to startup latency
    - Lots of inference deployers have proxies / gateways / higher levels in front of them (e.g. litellm, custom envoy proxies, model mesh in kserve) - is there anything we can do to make scale to zero work by default for most inference workloads (since inference is stateless and small and cheap to queue?)
    - What standard solutions (ingress, gateway) exist that can be used? Difference in gateway implementations limit reach, but extensions to servers might have standard components (e.g. ProxyWASM spec, https://github.com/ReToCode/envoy-request-buffer)
- Autoscaling reaction latency
    - Is the current reaction latency acceptable?
        - eg. 15s HPA re-sync loop time
        - plus any delays in metric scraping/serving

Problem space

- Request driven online inference as well as work-queue based near-realtime (O(<10m)) inference should be considered in scope

User Scenarios

Why does a user autoscale an inference workload? What is different for inference or accelerated inference vs regular workloads that makes autoscaling harder?

- Single workload
    - Autoscale to keep workloads at or under a per token latency target
    - Autoscale to keep accelerators at optimal latency/throughput
    - Autoscale to maximize throughput/accelerator
    - Autoscale an event queue driven inference workload
    - Autoscale a multi-host inference workload
- Density
    - Scale an accelerated inference workload to zero when total traffic is zero, so that other workloads can use its accelerator, and then hold traffic while it scales back up (requires startup to be fast enough to justify, and the ability to evict other workloads)
    - Allow a high priority inference workload to preempt batch inference when there are no accelerators available

Potential solution sharing

- Warm pooling to overcome pod startup challenges
    - Keeping nodes warm (provisioned)
    - Keeping pods warm (images pulled)
    - Keeping endpoints warm (queues / proxies)
- Improvements beyond request-based autoscaling for inference
    - Latency objective based autoscaling

Interesting papers

- https://arxiv.org/abs/2401.14351
-