2024년 09월 05일

AUTHOR

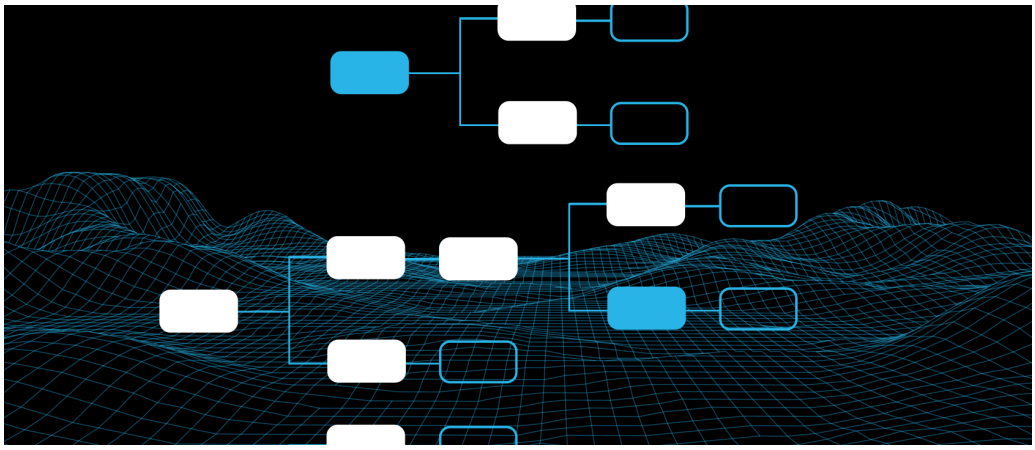
Vincent Chan

Hao Zhang

Flex Wang

Model Hotswapping: Optimizing AI Infrastructure and Enhancing LLM Efficiency

Gen AI



At Snowflake we support customer AI workloads by offering a diverse range of open source and proprietary large language models (LLMs). As we expand our first-party product offerings in Snowflake Cortex AI, including Cortex Analyst (in public preview), Snowflake Copilot (generally available in select regions) and Cortex Search (public preview), we have found that spinning up multiple dedicated instances for each new individual model can be costly and inefficient. In this article, we will explore model hotswapping, a technique for dynamically swapping LLMs on a static set of inference engines.

Previous model-scaling challenges

Although it might seem that inference is just about writing efficient kernel code, scaling our services to support thousands of customers required significant time and effort. We had to determine how to allocate our machines and how to support new models without increased infrastructure costs. Without the ability to swap models on the fly, we had dedicated pinned instances for each model. To determine the number of replicas for each model to deploy, we had to observe previous traffic patterns and reallocate model instances based on usage.

This resulted in a significant amount of manual work in determining how to allocate our machines to which models, along with releasing the set of model configurations into production, which took a non-trivial amount of time. Moreover, when we wanted to deploy new models to new regions, we commonly had to ask the question: Are there additional GPUs available for provisioning?

AUTHOR

Vincent Chan

Hao Zhang

Flex Wang

Why not only use traditional horizontal pod autoscaling (HPA)?

With the limited availability of high-end GPU resources from the different cloud providers supported by Snowflake, we reserve a fixed number of instances for availability. This practice contrasts with workloads that do not utilize GPU resources, where machines may be abundant and can horizontally scale up to an arbitrary amount of nodes. Because the number of GPU instances is fixed, the cost is also constant. Therefore, today we deploy our instances at their maximum capacity without additional instances to scale up to.

We run our LLM inference infrastructure across 42+ production clusters globally, each configured as a Kubernetes deployment. The easiest way to respond to a large workload or a burst of traffic for a given model is to horizontally autoscale the deployment for that model independently. To effectively utilize HPA, nodes need to be able to scale up quickly in response to real-time metrics. We can address the long cold start issue by keeping a set of reserve nodes hot with the image and model weights already loaded onto the node. This would make it very simple to scale up and down each different model deployment.

Unfortunately, this method was still not fast enough and inadequate for our use case. When scaling up and down pods on these hot nodes, we had to deal with different overheads of scheduling the pod, such as loading the model weights into GPU memory from disk for serving, and gracefully releasing resources upon scaling down. By the time the new pod is ready to serve production traffic, the current load may require an entirely different allocation of models! Very simply stated, our workloads were large enough and dynamic enough that HPA was too slow to respond to our services' needs. Knowing that HPA would continue to be a bottleneck and the number of models we support was likely to grow, we determined that in order to scale up/down resources in response to real-time traffic, what matters most is the speed of resource reallocation.

What does Model Hotswapping solve?

Efficiently serves a large set of models on fewer machines

Much like the name suggests, model hotswapping allows us to serve a large number of models on the same device. Unlike HPA, which allocates individual models to GPUs for active inference, our swappable inference engines contain many model weights from a large set of models in our registry. Since not all models can be loaded into GPU memory, model weights are cached in CPU memory and disk space.

AUTHOR

Vincent Chan

Hao Zhang

Flex Wang

Previously, scaling to meet load required scheduling and provisioning of a new specialized node. Now scaling only involves adjusting which models are **active**. When more replicas for a model are needed, the inference engine can quickly swap to one of the many models we offer.

Failover quickly

In production, there are sporadically instances when a node becomes unhealthy due to node failures and node pool upgrades. We specify a minimum number of replicas for each model that the model can drop down to. If the minimum number of replicas for a model is not met, we quickly identify a candidate model to swap in for failover.

Dynamic and efficient GPU resource utilization

Our LLM inference service offers more than 30 proprietary and open source models to be served in production today, including Snowflake Arctic, Jamba-Instruct, Mistral Large and Llama 3.1 405B. There are certain models receive little-to-no traffic, resulting in low GPU utilization for the machines they run on. By making model swapping responsive to workload demands, the inference engine can swap out models with low GPU utilization for those with higher workloads, making use of the GPU resources efficiently.

Model Hotswapping architecture

Define a weighted value for each model

At first, swapping models may seem simple: Create a list and swap the ones that are not used for those with longer queues. However, not every model has the same size and performance characteristics! We must take into account the speed performance of each model to make an informed model swapping decision. One of the simplest metrics you can use to measure performance and help determine a weight for each model is the throughput — models with higher throughput are assigned a lower weight and vice versa. It is possible that the workload for Model A equals the workload for Model B, yet Model B has 3x less throughput, so we may want to allocate more replicas for Model B to achieve equal throughput.

The algorithm is formalized as follows:
Define i as a unique model; p as the number of workloads for a given model; w as the weight for a given model (w indicates slowness of a model, larger w means the model need more GPU resource for a given workload) and r as the number of replicas for a given model. We calculate the load per replica for a model using:

AUTHOR

Vincent Chan

Hao Zhang

Flex Wang

Then we define t_1 to be our swapping threshold that is chosen by how much work our inference engine can handle at a time. To find the candidate model m_1 for swapping:

Then to find another candidate model m_2, to swap with m_1, we find the model with the least model replica load:

Define b as a scalar value for how big the difference from $f(m_1)$ needs to be from $f(m_2)$ in order to swap, and t_2 as a threshold for when models should not perform swaps (set just high enough so that models are not swapping at max capacity):

Loading model weights

Upon node startup, we download a set of model weights from centralized storage and load them into both CPU memory and disk memory. To speed up the time to swap, improving the time it takes to load models into GPU memory is critical. We preshard the model weights according to the tensor parallelism, such that each GPU only needs to read its respective sharded weights. This approach can improve model loading times significantly, decreasing them by 50% for most large models. In extreme cases, it brought the model loading time for Snowflake Arctic from 30 minutes down to just 2 minutes!

AUTHOR

Vincent Chan

Hao Zhang

Flex Wang

Figure 1. Memory hierarchy for time to load model weights into GPU. Loading from CPU memory is within the order of seconds; local disk is within the order of minutes; and blob storage much longer.

Ex e wapping endpoint from the inference engine

From the inference engine, we expose an endpoint that can be called to swap the current model with a requested model.

Upon receiving a swap request for a model:

1. The inference engine will evict all ongoing requests immediately or with a deadline, based on the urgency of the swap.
2. The engine will unload the current model from GPU memory back to CPU memory.
3. The engine will load the requested model from CPU memory or disk into GPU memory.
4. Start serving the requested model.

One decision we had to make was whether to let existing requests finish execution or evict them immediately with a deadline. Allowing existing requests to finish may take several minutes, depending on the model size, batch size and the number of total output tokens to be generated. We

decided against this since the executing requests may take some time to finish, resulting in our real-time decision to swap becoming more and more outdated as we wait for generation to finish.

To minimize the loss of work caused by evicting workloads for a given model in the middle of execution from an inference engine, the output tokens are streamed back from the inference engine to the client as they are produced and cached in memory. Since we want to route this request to another available inference engine, we send in the original input with the previously generated output tokens. Thus, the loss of work for evicting requests when swapping is the cost of prefilling the input tokens again along with the generated output tokens when rerouting requests after eviction.

Controller to gather metrics and make decisions to swap

With the rest of our inference deployment, we deploy a controller dedicated to manage model swaps. The control flow loop for the controller deployment is as follows:

1. Fetch metrics from the inference serving endpoint to determine the number of existing workloads for each model.
2. Fetch metrics from each inference engine to get the total number of 'ica r each model.
3. Weight each model's throughput with the current workload ato calculate the weighted workload per model replica.
4. Run through a list of rules; if a rule is violated, find a model with the least utilization that does not violate any rules when swapped.
5. If a suitable pair of candidates are found to swap in/out, make an asynchronous swap request to an inference engine with the model to be swapped out.

AUTHOR

Vincent Chan

Hao Zhang

Flex Wang

AUTHOR

Vincent Chan

Hao Zhang

Flex Wang

SHARE

Figure 2. Communication between the controller, API service and inference engines. Controller fetches metrics from the API service and makes a swapping decision, then selects an inference engine to swap.

Conclusion

By implementing model hotswapping in **Snowflake Cortex AI**, we can efficiently manage and scale our LLM infrastructure, ensuring optimal resource utilization and quick response to changing workloads. This technique allows us to dynamically swap models on a static set of inference engines, reducing the need for dedicated instances for each mc Th ot only cuts costs but also enhances the flexibility and responsiveness of our AI services. As we continue to expand our offerings, model hotswapping will be a key component in maintaining the efficiency and effectiveness of our LLM inference infrastructure.

SHARE

RELATED CONTENT

2024년 07월 23일

Achieve Low-Latency and High-Throughput Inference with Meta's Llama 3.1 405B using Snowflake's Optimized AI Stack

Meta's Llama 3.1 405B represents a groundbreaking milestone for open-weight large language models (LLMs), pushing the boundaries of what's possible in AI by bringing frontier model capabilities to everyone. However,...

More

2024년 06월 17일

Snowflake Arctic Cookbook Series: Instruction-Tuning Arctic

On April 24, we released Snowflake Arctic with a key goal in mind: to be...

Find Out How

2024년 08월 29일

Snowflake Cortex Analyst: Evaluating Text-to-SQL Accuracy for Real-World Business Intelligence Scenarios

Being able to build a system that allows business users to ask intricate, complex and...

Here's How

Flex Wang

SHARE

AUTHOR

Vincent Chan

Hao Zhang

Flex Wang

SHARE