AUGUST 2023

IT & DATA SCIENCE

Dynamic GPU Memory: Solving the Problem of Inefficient Resource Allocation in Inference Servers

Table of contents

Introduction 2
Fractional GPU with static memory allocation 2
Fractional GPU with dynamic memory allocation 3
Resolving memory collisions 3
Implementation in Kubernetes 4
Benefits of Dynamic GPU Memory 4
Conclusion 5

Introduction

In recent years, the demand for deploying inference servers and utilizing GPUs for various machine learning tasks has increased significantly. However, a common challenge arises when it comes to optimizing GPU memory usage for inference tasks. GPUs are becoming more powerful, equipped with larger memory capacities, but not all inference models require such extensive memory resources. This discrepancy leads to inefficient resource allocation, increased costs, and underutilization of GPU capabilities. In this blog, we will explore the motivation behind dynamic GPU memory, discuss the challenges it poses, and introduce how we at Run:ai address this problem.

Traditionally, when deploying models for inference, each model or inference server would occupy an entire GPU, regardless of the actual memory requirements. This approach is suboptimal because many models do not utilize the full memory capacity, resulting in wasted resources and increased costs. With modern GPUs having abundant memory, it becomes essential to find a way to leverage this excess capacity and improve GPU utilization.

Fractional GPU with static memory allocation

In Run:ai, fractional GPU refers to the practice of dividing a GPU into multiple logical partitions, each with GPU memory allocation. In the case of static allocation, the user needs to pre-configure the amount of GPU memory provisioned to each model or inference server.
This approach allows multiple inference servers to run on a single GPU, each with its own GPU memory allocation. However, the GPU memory consumption of a single inference server varies according to the input size and if the input size changes dynamically so is the GPU memory consumption.

For example, with dynamic batching, the GPU memory consumption is proportional to the instant batch size. For language models the GPU memory consumption depends on the length of the input sequence, for longer sequences the GPU memory consumption is larger. Fractional GPU with static memory allocation assumes a worst-case scenario, allocating the maximum possible memory requirements. This results in wasted GPU memory and higher costs.

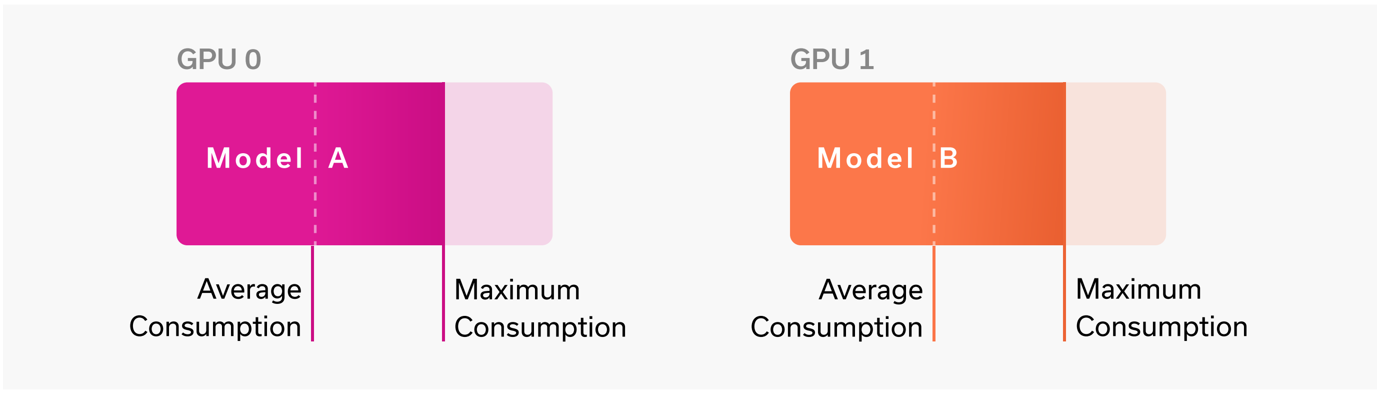

Figure 1. Models with dynamic GPU memory consumption

Fractional GPU with dynamic memory allocation

Run:ai introduces dynamic GPU memory allocation as a solution for inference servers with dynamic GPU memory consumption. Unlike traditional static memory allocation, dynamic allocation allows multiple models to share a single GPU while adapting to their varying memory requirements in real-time. This approach enables better utilization of GPU resources, reducing costs, and increasing GPU utilization. Dynamic GPU memory allocation works by pre-configuring the amount of GPU memory provisioned to each model or inference server as well as the amount of GPU memory to which the model can grow. This flexibility allows for on-demand adjustment of memory allocation, ensuring that models only use the memory they actually need for inference tasks.

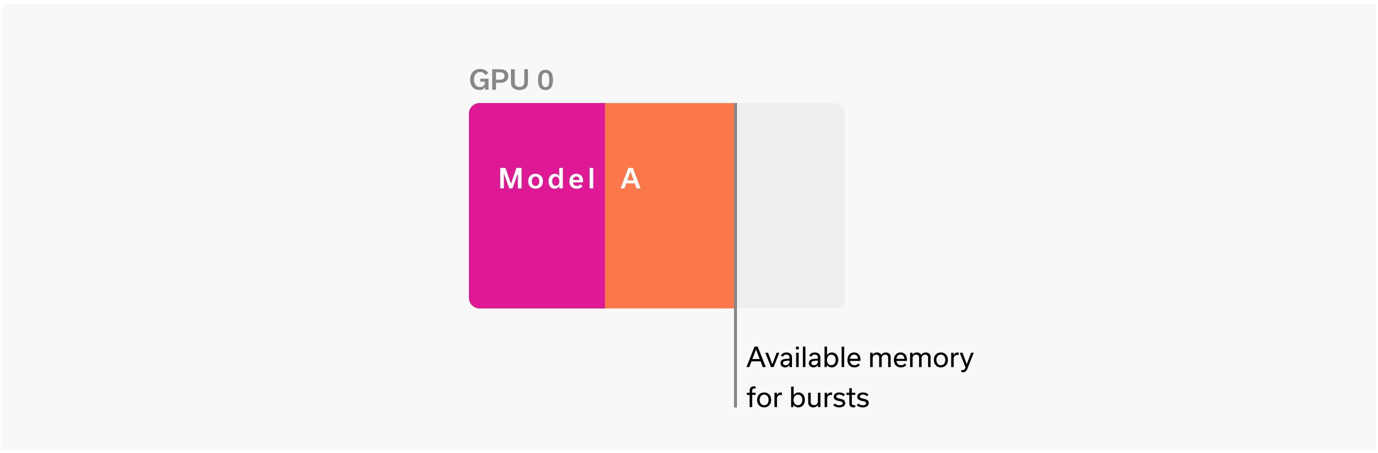

Figure 2. Deployment with dynamic GPU memory allocation

Resolving memory collisions

In case multiple inference servers simultaneously try to use burstable resources, causing a potential memory collision, the Run:ai system detects and resolves it in runtime. This ensures that memory collisions do not introduce OOM events which can lead to downtime and degraded SLA.

The system provides multiple memory collision resolvers which users can control:

1. A wait-and-retry resolver ensures that only one of the inference servers receives access to the GPU memory resources while the other servers wait until the resources get freed.
2. Priority resolver ensures that pods with higher priority will get burstable resources when needed, by terminating processes from other pods which are either idle for a configurable amount of time, or of a lower priority.
3. Swap Resolver allows the multiple servers to time-share the GPU memory by swapping memory between the CPU and GPU and ensuring consistency and fairness.

These memory collision resolvers provided by Run:ai allow users to handle memory collisions without requiring code changes or manual handling within their applications. Users have the flexibility to choose the appropriate resolver based on their specific requirements and preferences.

Benefits of Dynamic GPU Memory

Run:ai's fractional GPU with dynamic GPU memory allocations enables the following benefits:

1. Efficient Resource Allocation: Run:ai's dynamic GPU memory allocation ensures that models only utilize the necessary memory, preventing wastage of valuable resources.
2. Cost Savings: By running multiple models on a single GPU and dynamically adjusting memory allocation, organizations can significantly reduce costs by optimizing GPU utilization and avoiding unnecessary memory provisioning.
3. Increased Flexibility: With dynamic memory allocation, organizations can adapt to changing workload requirements without the need for manual reconfiguration, allowing for greater flexibility and scalability.

Dynamic GPU Memory with Run:ai

Implementation in Kubernetes

In Kubernetes, resource request and resource limit values are used to define the CPU and memory requirements of a containerized application.
The resource request value represents the amount of CPU and memory resources that a container expects to use on a node. It is used by the Kubernetes scheduler to make decisions about where to place the container. The request value is typically set based on the average resource usage of the application.
The resource limit value, on the other hand, sets an upper bound on the amount of CPU and memory that a container can use. It is used for resource allocation and to enforce resource isolation between containers running on the same node. If a container tries to exceed its limit, Kubernetes may take actions such as throttling or terminating the container.

Example of a K8s pod with request/limit for CPU and memory:

```
apiVersion: v1
kind: Pod
spec:
 containers:
 - name: app
   image: images.my-company.example/app:v4
   resources:
     requests:
       memory: "64Mi"
       cpu: "250m"
     limits:
       memory: "128Mi"
       cpu: "500m"
```

Run:ai's dynamic GPU memory capability allows users to specify different request and limit values for GPU memory, similar to how Kubernetes allows different request and limit values for CPU and memory.
This means that users can set a lower request value based on the average case and allocate more GPU memory up to the limit value when there is a larger input or the workload requires burstable resources.

Conclusion

Dynamic GPU memory allocation is a crucial solution to the problem of inefficient resource allocation in inference servers. With Run:ai's fractional GPUs and intelligent memory management, organizations can optimize GPU utilization, reduce costs, and improve performance of inference servers while maintaining the required SLA. By dynamically adjusting GPU memory allocation based on real-time usage and specific model requirements, Run:ai enables organizations to make the most of their GPU resources and streamline their machine learning operations. Embracing dynamic GPU memory allocation with Run:ai paves the way for more efficient and cost-effective deployment of inference models in production environments.

About Run:ai

Run:ai is an AI management platform for MLOps, Data Science, and DevOps teams. In addition to helping these teams access and utilize their GPU resources more effectively, it also has a powerful set of features that can abstract infrastructure complexities and simplify the process of training and deploying models. With or without a GPU shortage, Run:ai enables data scientists to focus on innovation without having to worry about resource limitations.

Read more about how Run:ai supports data scientists here

www.run.ai/runai-for-data-science