Efficient GPU Sharing for Serverless Workflows

Klaus Satzke, Istemi Ekin Akkus, Ruichuan Chen, Ivica Rimac, Manuel Stein, Andre Beck, Paarijaat Aditya, Manohar Vanga, Volker Hilt Nokia Bell Labs

Presented by

Divya Jacob
Nandhini Dhanasekaran
Nehaa Vuppala

Agenda

- Introduction
- The Problem
- Background
- Related works
- Knix Serverless Platform & Design
- Implementation
- Sandbox image flavours
- Evaluations
- Strengths and weaknesses
- Conclusion

Introduction

- Serverless computing is a cloud computing execution model in which the cloud provider allocates machine resources on demand, taking care of the servers on behalf of their customers.
- The function development environment of all serverless computing frameworks at present is CPU-based
- This paper proposes to extend the open-sourced KNIX high-performance serverless framework so that it can execute functions on shared GPU cluster resources
- KNIX is an open source serverless computing platform for Knative or virtual machine-based environments. It considerably increases resource efficiency and reduces function startup overhead.

Deep Learning Application Example

- Consider an image processing pipeline for Neural Style Transfer operation where a number of consecutive functions such as pre-processing, processing and post-processing are executed
- The application was run on different deployments of the KNIX platform
- The total execution time is more when omitting accelerators.

Why is it a problem?

Deep Learning Application Example

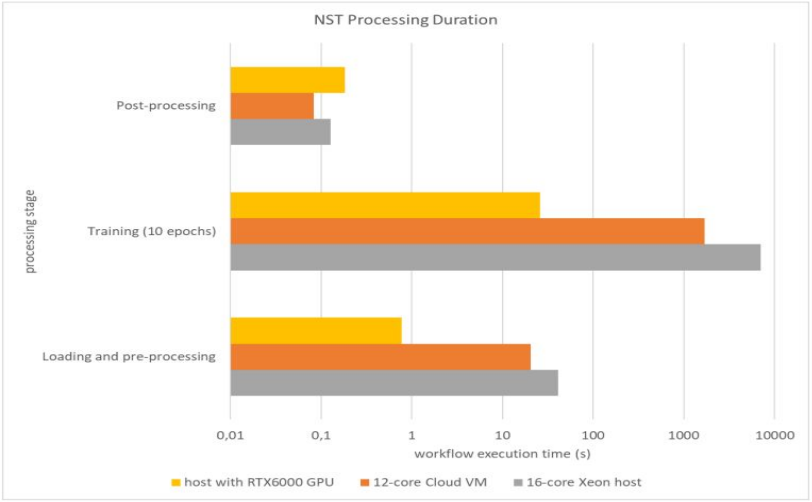

Figure 1: Execution duration of Neural Style Transfer Serverless Workflow on different KNIX deployments

Background

Serverless Computing

- AWS Lambda as well as other FaaS public services imposes strict computing requirements based on execution time, RAM and disk storage
- It is not possible to assign any kind of accelerator to a FaaS

Accelerated Serverless Computing

- Kubernetes provides the capability to schedule NVidia GPU containers, but it is generally implemented by allocating an entire GPU card to a container and only allows to schedule entire GPUs to pods

Motivation

- In a typical ML pipeline predictions or inference accounts for up to 90% of total operational costs. In order to improve the GPU utilisation there is a need to allow more inference tasks to share the same GPU card.
- For fine-grained GPU device scheduling, there is currently no good solution. This is because extended resources such as GPU in Kubernetes does not support the allocation of complex resources and limits the quantity of extended resources to whole integers.
- Using the KNIX serverless platform, the user can arrange any function to use just a portion of a GPU core and memory, in contrast to comparable methods.

Related Work

1. **GPU attachment to CPU service**

   Approach is to integrate the open source serverless computing framework with NVIDIA-docker and deploy services based on the GPU support container in order to deploy services faster than existing serverless computing framework

   If the user wants to use both CPU and GPU simultaneously, this is not possible with the GPU attachments approach

2. **CUDA Without Environment Setup**

   Enables remote GPU based code acceleration.

   Disadvantage is that it lacks fault tolerance in case of network outages.

Related Work

3. GPU Sharing Frameworks

When a GPU device cannot be fully utilized by a single program, there is low resource utilisation if support for GPU sharing between multiple concurrent containers is absent.

Few examples that support GPU sharing are KubeShare, GPUShare Scheduler Extender, GPU Manager Framework.

KNIX SERVERLESS PLATFORM

- The authors test system implementation uses the serverless framework KNIX which was earlier named as SAND that aims at low latency and efficient utilization of resources.
- KNIX microfunctions is an open-source serverless computing platform for KNative or virtual machine based environments.
- The KNIX functions combine container-based resource isolation with a light-weight and process-based execution model that provides significant improvements in resource efficiency and reduces the startup overhead for functions when compared to SAND framework .

- The Python KNIX functions are executed in sandbox using NVidia GPU resources for helm deployment.
- The GPU nodes are detected and configured by the serverless platform automatically.
- The GPU manager manages the NVidia GPU devices in our Kubernetes test cluster as it permits flexible GPU memory sharing as well as computing resources among several isolated containers.
- This is done by partitioning the physical GPU's into multiple virtual GPU's by assigning one kubernetes pod with virtual GPUs(vGPUs) as requested.

KNIX PLATFORM DESIGN

The goal of KNIX framework is to share both GPU memory and compute resources among containers with maximum performance enhancement but with minimum expenses.

CHALLENGES FACED:

- **Transparency**
- **Low overhead**
- **Isolation**

IMPLEMENTATION OF KNIX PLATFORM

ARCHITECTURE AND MODIFICATIONS OF PLATFORM FOR GPU SHARING:

- KNIX management service(MS) - Functions that allows users to sign up, login and manage the micro functions and the workflows because the Management Service accesses its own storage .
- For deployment on Kubernetes, the MS should also consider the KNative service configurations and parameters.
- This requires the extension of KNIX MS into several parts.
- KNIX Graphical User Interface or KNIX Software Development Kit- Provides configurations for modifying micro functions for deployments on sandbox.

- The Helm deployment allows to define resources in the form of requests and limits to control CPU and memory however this technique should be extended in order to address virtual GPU resources for a micro function.
- The MS will analyse the complete workflow of this mechanism for added information of GPU requests and limits.
- For multiple micro functions with GPU requirements, the MS will sum up all their requirements to configure the KNative service.
- Before each workflow deployment,the final logic will be appended to the MS for any queries on the capacities of virtual GPUs that can be allocated in the cluster so that the MS can detect if the virtual GPUs required by the workflow are present in the cluster to avoid over subscription of pending workflows.

SANDBOX IMAGE FLAVOURS

- New GPU image flavours need to be created in addition to existing flavours for KNIX supported languages.
- Required to provide the Deep Neural Network libraries,math and CUDA tools to address the GPU within the container.
- To run this flavour, we need to install container toolkit of Nvidia and docker run time on the GPU nodes of the cluster.

TESTBED

What is it about?

In order to obtain a wide range of performance data for virtual Environment interaction techniques, formal frameworks and experiments are used in the testbed evaluation.

Vagrant and Kubespray

- Cluster (built) —----------> Five hosts.
- One host acts as the Kubernetes master.
- Remaining nodes act as worker nodes of which one of them has a physical GPU.
- Kubernetes master - 8 CPUs and 32 GB of RAM.
- Worker nodes-
- CPU configuration-16x Intel(R) Xeon(R) W-2245 CPU
- GPU configuration-NVidia Quadro RTX 6000
- RAM -128GB

Issues related to sharing GPUs for serverless functions in KNIX:

- Overhead
- Application performance
- Resource Isolation
- Dynamic Sandbox Resource Allocation
- Fast Data Sharing

Overhead

- We build and deploy the GPU manager device plugin as a daemonset which ensures that all or some nodes run a copy of a pod in the cluster.
- The performance of GPU applications that are deployed in KNIX sandboxes are evaluated.
- Deep Learning technique is employed in building GPU applications as they provide an easy way to construct deep networks.

- When GPUs are shared among sandboxes, varied overhead is found due to different libraries and resource allocation strategies of different Deep Learning frameworks.

- Popular Deep Learning Frameworks such as Tensorflow MNIST , Pytorch MNIST and MXNET MNIST are selected for our test.

- MNIST- A program that detects handwritten digits with database using Convolutional Neural Network(CNN).

- This application is run on these three different Deep Learning frameworks in both bare metal and inside the KNIX sandbox to measure it's execution time.

Definition of notations:

- T_baremetal : execution time of the test application running on the host with full GPU assigned to it.
- T_vGPU : execution time of the applications running in a sandbox with shared GPUs
- Overhead is the difference between T_baremetal and T_vGPU:

$$Overhead = \frac{t_{vGPU} - t_{baremetal}}{t_{baremetal}} * 100\%$$

- All tests are repeated 10 times and the average of them is used.

- It is observed that the measured overhead of the GPU manager in case of Tensorflow test is quite low and hence can support performance to be achieved for the KNIX platform.
- With Torch and MXNet frameworks, the overheads are quite large of almost 15% which is related to large fluctuations in the GPU utility functions while no such fluctuations were observed during measurement with Tensorflow.
- This allowed for more efficient allocation of vGPU resources by the GPU manager for Tensorflow framework.

Table 2: Measured GPU virtualisation overhead

	$t_{baremetal}$ (s)	t_{vGPU} (s)	Overhead (%)
Tensorflow MNIST	298.2	300.0	0.67
Pytorch MNIST	189.1	222.3	14.9
MXNET MNIST	39.32	33.50	14.8

APPLICATION PERFORMANCE

Table 3: Configurations of vGPUs for application performance experiments

Number of vGPUs	1	2	4	6	8
Comp. resource per vGPU (%)	100	50	25	15	12.5
Memory per vGPU (GB)	2	2	2	2	2

- This evaluation is about the impact of partitioning of GPU resources on application performance.
- The physical GPU is partitioned into 1,2,4,6 and 8 virtual GPUs and are assigned to KNIX sandboxes each executing the same DL application.

- The execution time is linear to the computing resources of virtual GPU for a large number of concurrently executing pods
- However for small number of pods,the execution time is non-linear especially for applications that handled large model sizes.

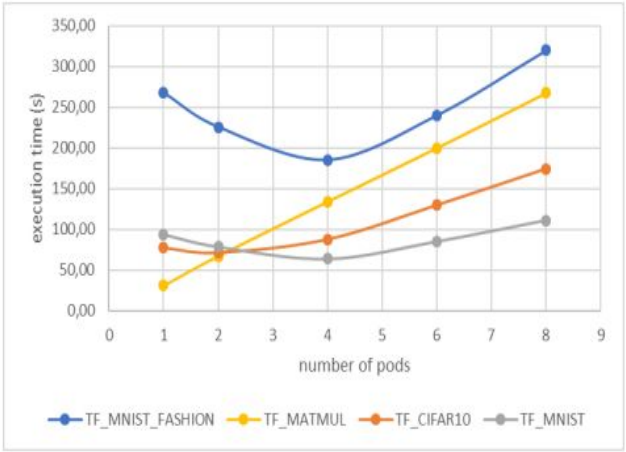

Figure 3: The performance of DL test applications under different partitioning of computing resources

Resource Isolation

- Assigning resources to one sandbox should not unintentionally impact other sandboxes.
- On one physical GPU, we launch 2, 4, and 8 sandboxes and measure their memory and GPU utilization.

Table 4: Configurations of vGPUs for isolation experiments

Number of vGPUs	2	4	8
Computing Resource per vGPU	50%	25%	12.5%
GPU memory per vGPU (GB)	7.68	3.84	1.54

Dynamic Sandbox Resource Allocation

- The GPU Manager approach offers the special feature of elastic GPU resource allocation
- Using one physical GPU, we ran two workflow sandboxes simultaneously on KNIX sandboxes, executing concurrently.
- The execution time of the workflow with 0.1 GPUs is reduced by 80% compared to the hard resource limit case with static resource allocation, and it is slightly increased by 15% with 0.9 GPUs.

Experimental results indicate that elastic resource allocation significantly improves application performance and GPU utilisation, especially when low resource allocation values are used.

During this experiment, the average GPU utilization of the physical GPU increased by 73.5%.

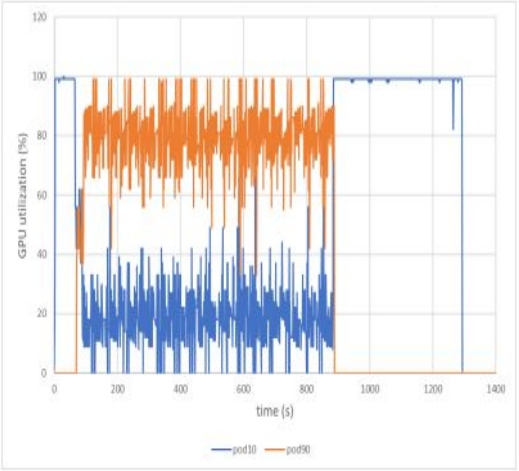

Figure 6: Dynamic GPU resource allocation dynamics

Fast Data Sharing

Characteristics: (Persistent Volumes (PVs) and Persistent Volume Claims (PVCs))

- If a Pod is destroyed, data is persisted.
- A PV can be used to share data between sandboxes, whether in the same pods or across pods.
- A PV does not allow to share data between pods running on different nodes.
- PVs can be accessed by many Pods at the same time.

Example

- There is an interesting alternative to share data among functions offered by KNIX.
- Micro functions belonging to the same workflow could be configured to share GPU sandbox resources and exchange model data via the local file system in a fast and efficient manner instead of running each GPU-based inference function in its own sandbox.
- Here, the author prepared KNIX sandboxes and have evaluated Keras model data loading latencies using two different scenarios:

    1. Data exchange via the GPU-using Sandbox filesystem
    2. Via PV/PVC for sandboxes on the same GPU node

It has been shown that when sharing Keras VGG19 model and weight data via reading/writing to a local file system,the time required for loading the same VGG19 model data into GPU memory via PV/PVC is slightly shorter in all measurements,showing a small advantage of 14-26% depending on the data loading scheme.

Thus KNIX offers greater benefits through the grouping of functions inside the same KNIX sandbox, since direct data exchange via the sandbox file system allows for a faster model sharing process.

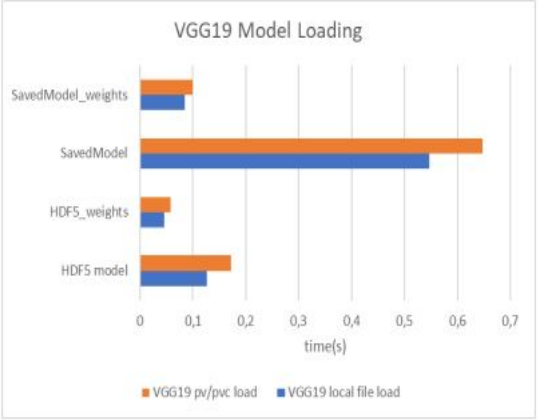

Figure 7: Comparison of read/write speeds of GPU-using KNIX sandboxes

Strengths and Weaknesses

Strengths:

- The paper explains in detail about the traditional methods of processing GPU workloads on Kubernetes cluster versus KNIX method of dealing with this issue.
- Experimental studies have been conducted to evaluate the performance.

Weaknesses:

- Vagrant and kubespray are not briefed clearly in the paper for the creation of five hosts.
- The paper has a spelling error.
- Keras VGG19 model was not mentioned in the paper.

Conclusion

- An approach for sharing GPU memory and computing resources using KNIX serverless platform was implemented. KNIX performance is high when compared with AWS Lambda.
- Successfully partitioned the physical GPU into 1,2,4,6, and 8 vGPUs, and assign them to serverless KNIX workflows and microfunctions and executed concurrently.
- Researchers have evaluated the performance impacts and overheads of GPU sharing using different DL frameworks.

Thank you!!

Questions?