Parallelizing machine learning application in the cloud with Kubernetes: A case study

Rajy Rawther, Advanced Micro Devices
September 2020
Agenda

• ML application end-to-end flow
• Kubernetes: A brief introduction
• Scaling image classification in the cloud using Kubernetes
• Scaling inference from one server node to many
• Load balancing and identifying bottlenecks in the deployment pipeline
• How to choose between CPUs or GPUs for performance
• Conclusion
  • Kubernetes is available at https://github.com/kubernetes/kubernetes and licensed under Apache 2.0
ML application end-to-end flow

- CPUs/GPUs or dedicated
- With right software infrastructure

- Data collection is tedious and quality of data matters
- Need synthetic data generation with tools for augmentation

- Start with simple model, train and update for accuracy
- Need many CPU and GPU cores to run these in a short amount of time

- Efficient data pipeline is required to get optimal throughput
- When the model is given new data, there is a need to evaluate and update the model for accuracy.
## ML Software stack for application deployment

### Applications
- Vision
- NLP
- Classification/Detection
- Video

### Cluster Deployments
- Docker
- Kubernetes
- Singularity
- SLURM

### Frameworks and exchange formats
- TensorFlow
- PyTorch
- Caffe2
- ONNX
- NNEF

### Middleware libs
- Blas/Eigen
- RNG/FFT
- MIOpen
- CuDNN

### Programming models
- OpenCL
- HIP
- Cuda
- OpenVX
- Python

### Processors
- CPU
- GPU
- APU
- DLA
Kubernetes® (K8s): A brief introduction

Kubernetes is an open-source system for automatic deployment, scaling and management of containerized applications

- **Node**
  - Runs Kubelets (“node agent” service)
  - Communicates with master
  - Runs Pods

- **Pod**
  - Runs one or more containers
  - Exists on a node

- **Service**
  - Handles requests
  - Load balances

- **Deployment**
  - Defines what you want (cluster services); Kubernetes handles it for you
Need for containerized and scalable deployment

Each deployment is self-contained & can run on any system which has the hardware resources.

Easily manage resources over multiple servers

The ability to cluster and schedule container processing to scale

Can easily access data across the nodes

The Task

Develop the application

Create docker container

Launch on Kubernetes
Typical deployment YAML file for Kubernetes configuration

```
# deployment yaml file
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mivisionx-deployment
  labels:
    app: mivisionx-server
spec:
  replicas: n #number of replicas
  selector:
    matchLabels:
      app: mivisionx-server
  template:
    # define the pods specifications
    metadata:
      labels:
        app: mivisionx-server
    spec:
      containers:
        - name: mivisionx-server
          image: mivisionx/ubuntu-18.04:rocm3.3
          ports:
            - containerPort: 28282
          workingDir: /root
          env:
            - name: HIP_VISIBLE_DEVICES
              value: "0" # # 0,1,2,...,n for GPU, -1 for CPU
          command: ["/bin/sh", "-c", "--"]
          resources:
            limits:
              amd.com/gpu: 1 # requesting 1 GPU
```

Service YAML configuration

```
---
apiVersion: v1
kind: Service
metadata:
  name: mivisionx-deployment-service
  namespace: default
spec:
  type: NodePort
  selector:
    app: mivisionx-server
  ports:
    - port: 28282 #port accessible inside the cluster
targetPort: 28282 #port which sends traffic from service to container
    nodePort: 30001 #port which is accessible outside the cluster
    protocol: TCP
    labels:
      app: mivisionx-server
```
Load balancing a deployment pipeline with CPU and GPU

Model Initialization is done only once as part of initialization.

Preprocessing involves decoding and applying many transformations.

CPUs are best suited for preprocessing and postprocessing.

Data-parallel processors are efficient for running inference on a batch of images.

Postprocessing is needed to produce useful result.
Inference deployment client server application case study

1. Choose model & parameters
2. Choose dataset
3. View results

Critical path flow

Model and Parameters
Status
Setup Phase

Initialize Inference
(Compile, build inference graph, set-up hardware)

Up to 8 GPUs on a single server node

Image database
Results

Client Application

A
G

CPU cores

D

G

GPU #0
GPU #1
GPU #2
GPU #3
...

E

Multi-GPU inference

F

Images
Results

Inference Execution

B

G

Resume model

C

Images

Image decode

Image Transform

© 2020 Advanced Micro Devices
Inference Deployment Using Kubernetes

Client Desktop/Server

Inference Client Application

Kubernetes

K8s Load Balancer

Server

K8s POD

Allocated CPU & GPU

Container mapped to K8s Pods

Application image

Inference Server Container

© 2020 Advanced Micro Devices
Scaling ML Inference with Kubernetes®

ML Inference

24 Kubernetes® Pods
Accelerated by 24x GPUs

Node = 8 pods
Pod = 1 GPU + 8 CPU
Performance Graph with multiple GPUs and CPUs

98% scaling
Steps for achieving linear scaling

• Remove **bottlenecks** in the inference server critical path.

• Allocate **hardware resources** for each deployment pod. In this case we choose **1 GPU and 8 CPU cores**

• The application needs to maintain **separate queues** (as shown in the next slide) for each instance of application so multiple instances won’t block each other.

• The model is **pre-launched and initialized** for each nod separately

• The **data loading bottleneck** is avoided by **preloading input images for each nod** in advance

• Finally, use a **multi-threaded client application** that feeds and sends requests for each of the K8s pod with minimal latency.
Inference server critical path

CLIENT: READ HDD → CLIENT: XMIT → SERVER: JPEG DECODE → COPY: PCIE TO GPU → GPU: INFERENC → SERVER: SEND RESULTS → CLIENT: DISPLAY RESULTS

Limiting factor or tasks

A - G Critical path flow

Drive Speed
NVMe = ~3.5GB/sec (~5000fps)

Network bandwidth
~100 Gbps

JPEG decode, resize and convert to tensor multi-threaded
~200 MB/sec *128 threads (~100000 fps)

PCle™ bandwidth
32GB/s for x16 (~50000 FPS)

Execute batched deep learning inference on the model
1200 fps (fp16 on MI50)

Collect inference results (PCle™ from GPU to CPU) and xmit back to client
~32 GB/s for x16

Check and show results
Update performance specs
Server processing queues

- **Input Queue**
  - One master input queue

- **CPU**
  - One queue per GPU

- **CPU cores**
  - Decode
  - Resize
  - Convert to tensor

- **Pre-processing queues**
  - GPU #1
  - GPU #2

- **Inference queues**
  - GPU #1
  - GPU #2
  - Inference on Model (ResNet50)
  - Predictions
  - Convert to labels

- **Retirement queues**
  - GPU #1
  - GPU #2
  - Predictions

**TCP/IP**
Overall balance of different stages of pipeline

Classification pipeline stages, potential FPS

Overall FPS

<table>
<thead>
<tr>
<th>Series1</th>
<th>Series2</th>
<th>Series3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>32</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
Kubernetes Pros and Cons

**PROS**
- Deep integration into cloud native ecosystem
- Broad support for containers and runtimes
- Automatic scaling and load balancing
- Efficient resource management
- Multiple workloads and deployment options
- Built-in security
- Integration with major cloud providers

**CONS**
- Steep learning curve
- Challenging to install and configure manually
- Not suited for simple applications and can reduce productivity adopting it
- Need expensive talent to adopt it
Next Steps: Collecting metrics and evaluating the results

• Each K8s container can collect metrics asynchronously.
• Evaluating the neural network model guarantees the model will perform well given new data.
• Various tools can be used to evaluate model for a given dataset.
• E.g., inference analyzer to validate different models on one or more dataset.
Final Thoughts

- Intelligent load balancing between CPU and GPU can increase the overall throughput.
- By carefully analyzing the input pipeline and identifying parts that affect the gross performance, the latency can be minimized.
- Different image sizes and complex models tip the workload to weigh on CPU/GPU or data-parallel processors.
- Maintaining data queues at various stages is essential to reduce data-transfer bottlenecks.
- Smaller batch sizes lower latency but doesn’t give the best performance.
- Finally, having the right tools to visually analyze the results is key to understand the whole picture.
As ML takes over many parts of our lives from protecting people to autonomous driving, we need simple automated ways to deploy and scale those applications.

Data scientists need to analyze and iterate data-sets, algorithms and without slowing them down or placing heavy burden on company resources.

By carefully developing your application for cloud-native environment with containers and micro-services, scientists can greatly scale those for portability and performance.
References

MIVisionX

Kubernetes
https://kubernetes.io/docs/home/

ResNet
https://github.com/onnx/models/tree/master/vision/classification/resnet
Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED ‘AS IS.” AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

© 2020 Advanced Micro Devices, Inc. All rights reserved.

AMD, the AMD Arrow logo, Epyc, Radeon, ROCm and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.