

Accelerated Apache Spark 3 leveraging GPUs on VMware Cloud

Table of Contents:

Accelerated Apache Spark 3 leveraging GPUs on VMware Cloud	1
1 Introduction.....	4
2 Apache Spark	4
2.1 Spark 3 adds GPU Awareness	4
2.2 Spark 3 provides enhanced support for Deep Learning	4
2.3 Better Kubernetes Integration	4
2.4 Spark 3.0 with Kubernetes operator:	5
2.5 Accelerated Analytics and AI on Spark	5
2.6 NVIDIA RAPIDS:	6
2.7 New RAPIDS Accelerator for Spark 3.0	6
2.8 XGBOOST:.....	6
3 Other Components of the Solution:.....	7
3.1 Spark History server:	7
3.2 Harbor Container Registry:	7
3.3 VMware Cloud Foundation with Tanzu	7
3.4 Solution Components:.....	8
3.5 Building Blocks of the solution:	8
4 Use Case 1: TPC-DS with NVIDIA RAPIDS	8
Deployment steps:.....	9
4.1 Installation of Spark operator for Kubernetes.....	9
4.2 Install spark history server	9
4.3 Creation of docker image to run tpcds application related tasks	9
4.4 Build TPC-DS scala distribution package	10
4.5 Install and configure Harbor	10
4.6 TPC-DS Data generation	11
4.7 Configure Kubernetes worker nodes for running GPU applications	11
4.8 TPC-DS validation testing	12
4.9 TPC-DS Results	12
5 Use Case 2: Model mortgage data using Spark XGBoost running on Kubernetes with Spark Operator.....	12
5.1 Use Case Prerequisites	13
5.2 Create docker image to run Spark XGBoost application related tasks	13
5.3 Data leveraged in use case	13
5.4 Build Scala distribution jars for Mortgage ML.....	13
5.5 ETL conversion of Mortgage data	14
5.6 Mortgage ML training.....	14
5.7 Mortgage ML test	14
6 Conclusion	15
7 Appendix A.....	16
8 Appendix B.....	18

9	Appendix C.....	19
10	Appendix D.....	20
11	Appendix E.....	21
12	Appendix F.....	23
13	Appendix G.....	24
14	Appendix H.....	25

1 Introduction

Apache Spark is a unified analytics engine for large-scale data processing. The recent release of Apache Spark 3.0 includes enhanced support for accelerators like GPUs and for Kubernetes as the scheduler. VMware Cloud Foundation 4.x supports Kubernetes via Tanzu and provides enhanced accelerator capabilities. VMware Cloud Foundation can be a great platform for Apache Spark 3 as it supports the new capabilities of GPU acceleration and Kubernetes. This solution seeks to validate the VCF platform with Tanzu for Apache Spark 3. NVIDIA RAPIDS and XGBOOST are important components that are optimized for Apache Spark 3 and our solution will validate these use cases on the VMware Cloud platform.

2 Apache Spark

Apache Spark is the de facto unified engine for big data processing, data science, machine learning and data analytics workloads. This year is Spark's 10-year anniversary as an open source project. Since its initial release in 2010, Spark has grown to be one of the most active open source projects.

Recently released Apache Spark 3 adds compelling features like adaptive query execution; dynamic partition pruning; ANSI SQL compliance; significant improvements in pandas APIs; new UI for structured streaming; up to 40x speedups for calling R user-defined functions; accelerator-aware scheduler; and SQL reference documentation.

2.1 Spark 3 adds GPU Awareness

GPUs and other accelerators are widely used for accelerating specialized workloads like deep learning and HPC applications. While Apache Spark is used to process large datasets and complex data scenarios like streaming, GPUs that are needed for machine learning by data scientists were not supported until recently. Spark did not have awareness of GPUs exposed to it and was not able to request GPUs and schedule them for users causing a critical gap for the unification of big data and AI workloads.

Spark 3 bridges the gap between Big Data and AI workloads by

- Updating cluster managers to include GPU support and exposing user interfaces to allow for GPU requests
- Updating the scheduler to understand availability of GPUs that are allocated to executors, user task requests, and assign GPUs to tasks properly.

2.2 Spark 3 provides enhanced support for Deep Learning

Deep Learning on Spark was possible in earlier versions, but Spark MLlib was not focused on Deep Learning, its algorithms and didn't offer much support for image processing. Hybrid projects like TensorFlowOnSpark, MMLSpark, etc. made it possible but using them presented significant challenges. Spark 3.0 handles the above challenges much better with its added support for accelerators from NVIDIA, AMD, Intel, etc. In Spark 3.0 vectorized UDFs can leverage GPUs for acceleration.

2.3 Better Kubernetes Integration

Spark support for Kubernetes was rudimentary in the 2.x version as it was difficult to use in production. Its performance was lacking when compared to that of the YARN cluster manager. Spark 3.0 introduces the new shuffle service for Spark on Kubernetes that allows dynamic scale up and down of Spark executors in Kubernetes.



Figure 1: GPU accelerated GPU Accelerated Apache Spark 3

Spark 3.0 also supports GPUs with Kubernetes providing for pod level isolation of executors, making scheduling flexible on a GPU enabled cluster.

2.4 Spark 3.0 with Kubernetes operator:

The Kubernetes Operator for Apache Spark makes running [Spark](#) applications as easy and seamless as running other workloads on Kubernetes. The Kubernetes Operator for Apache Spark ships with a command-line tool called `sparkctl` that offers additional features beyond what `kubectl` is able to do. It uses [Kubernetes custom resources](#) for specifying, running, and surfacing the status of Spark applications.

2.5 Accelerated Analytics and AI on Spark

Spark 3.0 marks a key milestone for analytics and AI, as ETL operations are now accelerated while ML and DL applications leverage the same GPU infrastructure. The complete stack for this accelerated data science pipeline is shown below. The use cases for this solution will leverage RAPIDS and XGBoost in this stack to validate the capabilities of VMware Cloud Foundation.

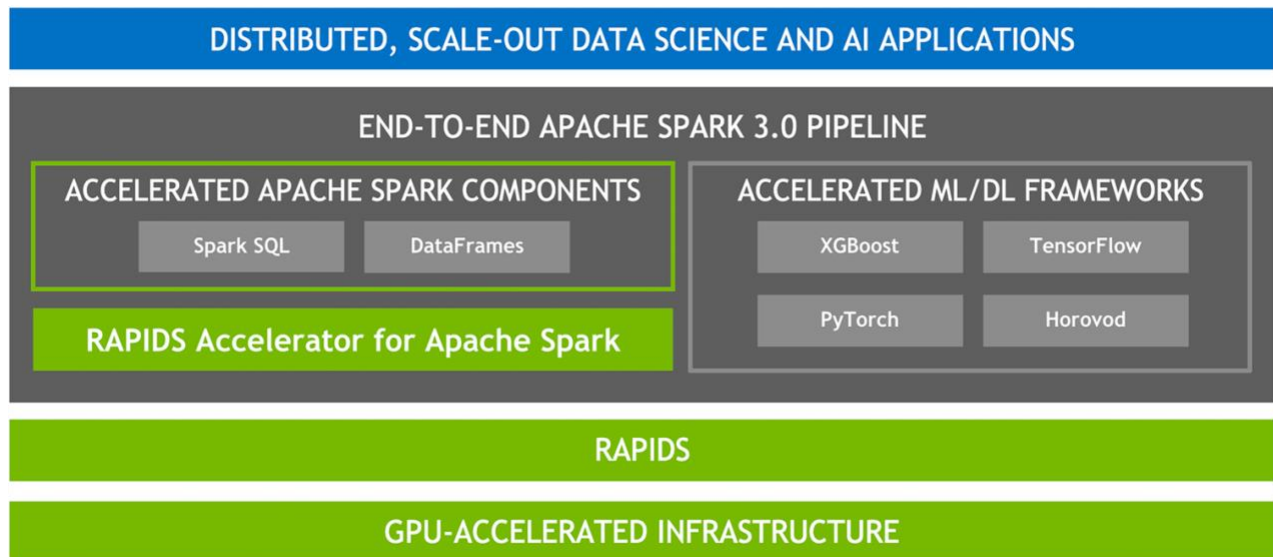


Figure 2: GPU accelerated GPU Accelerated Apache Spark 3. (Source: [GPU Accelerated Apache Spark](#))

2.6 NVIDIA RAPIDS:

The RAPIDS suite of software libraries, built on [CUDA-X AI](#), gives you the freedom to execute end-to-end data science and analytics pipelines entirely on GPUs. It relies on NVIDIA® CUDA® primitives for low-level compute optimization but exposes that GPU parallelism and high-bandwidth memory speed through user-friendly Python interfaces. RAPIDS also focuses on common data preparation tasks for analytics and data science. This includes a familiar DataFrame API that integrates with a variety of machine learning algorithms for end-to-end pipeline accelerations without paying typical serialization costs. RAPIDS also includes support for multi-node, multi-GPU deployments, enabling vastly accelerated processing and training on much larger dataset sizes.

2.7 New RAPIDS Accelerator for Spark 3.0

[NVIDIA CUDA](#)® is a revolutionary parallel computing architecture that supports accelerating computational operations on the NVIDIA GPU architecture. NVIDIA has created a RAPIDS Accelerator for Spark 3.0 that intercepts and accelerates ETL pipelines by dramatically improving the performance of Spark SQL and DataFrame operations.

RAPIDS offers a powerful GPU DataFrame based on Apache Arrow data structures. Arrow specifies a standardized, language-independent, columnar memory format, optimized for data locality, to accelerate analytical processing performance on modern CPUs or GPUs. With the GPU DataFrame, batches of column values from multiple records take advantage of modern GPU designs and accelerate reading, queries, and writing. (Additional details at [Acceleration Apache Spark with GPUs](#))

2.8 XGBOOST:

XGBoost is a well-known gradient boosted decision trees (GBDT) machine learning package used to tackle regression, classification, and ranking problems. It's written in C++ and NVIDIA CUDA® with wrappers for Python, R, Java, Julia, and several other popular languages. XGBoost now includes seamless, drop-in GPU acceleration, which significantly speeds up model training and improves accuracy for better predictions.

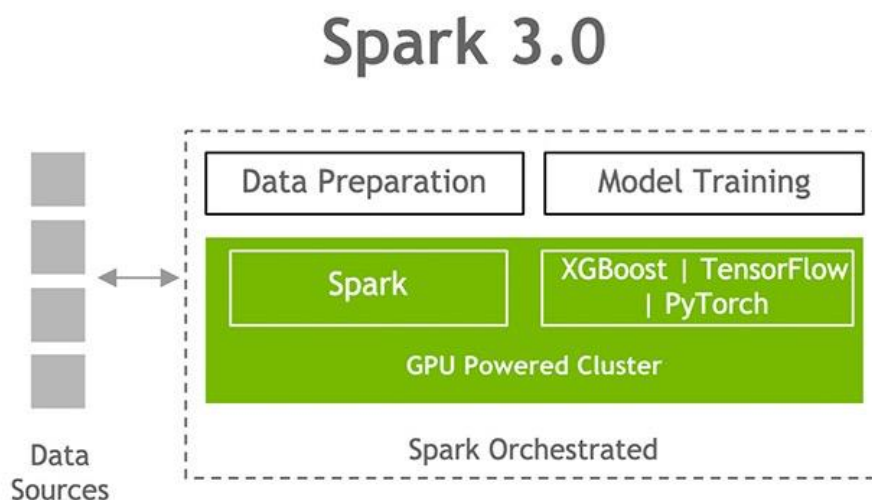


Figure 3: XGBOOST for GPU powered Apache Spark 3 (Source: [GPU Accelerated Apache Spark](#))

3 Other Components of the Solution:

3.1 Spark History server:

The Spark History Server is a User Interface that is used to monitor the metrics and performance of the completed Spark applications. This is where Spark history Server comes into the picture, where it keeps the history (event logs) of all completed applications and its runtime information which allows you to review metrics and monitor the application later in time. History metrics are very helpful when you are trying to [improve the performance of the application](#) where you can compare the previous runs metrics with the latest run.

3.2 Harbor Container Registry:

Harbor is an open source trusted cloud native registry project that stores, signs, and scans content. Harbor secures artifacts with policies and role-based access control, ensures images are scanned and free from vulnerabilities, and signs images as trusted. Harbor extends the open source Docker Distribution by adding the functionalities usually required by users such as security, identity and management. Having a registry closer to the build and run environment can improve the image transfer efficiency. Harbor supports replication of images between registries, and also offers advanced security features such as user management, access control and activity auditing. Harbor is a CNCF Graduated project that securely manage artifacts across cloud native compute platforms like Kubernetes and Docker.

3.3 VMware Cloud Foundation with Tanzu

VMware Cloud Foundation with Tanzu delivers hyper-speed Kubernetes that provides agility, flexibility and security for modern apps. VMware Tanzu delivers the infrastructure, and services to meet changing business needs to rapidly deploy new applications. VCF provides consistent infrastructure and operations with cloud agility, scale and simplicity.

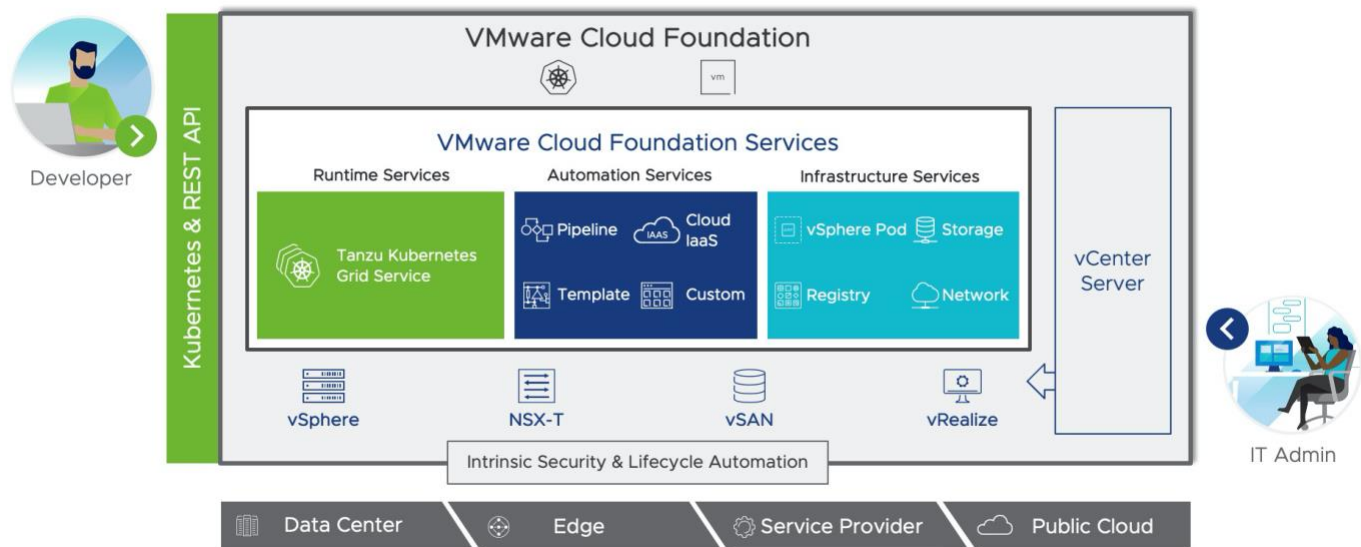


Figure 4: VMware Cloud Foundation with Tanzu

VMware Cloud Foundation with Tanzu is a Hybrid Cloud Platform that accelerates development of modern applications that automates the deployment and lifecycle management of complex Kubernetes environments.

- IT admins have complete visibility and control of virtualized compute, network and storage infrastructure resources through VCF.
- Software defined compute, storage and networking with vSphere, NSX-T and vSAN/VVOL provides ease of deployment and automation.
- Developers have frictionless access to Kubernetes environments and infrastructure resources through VCF Services.
- VMware Cloud Foundation provides runtime services automation services and infrastructure Services, all delivered via Kubernetes and RESTful APIs

3.4 Solution Components:

There are two distinct use cases that were deployed and validated in the solution. These include

1. TPC-DS with NVIDIA RAPIDS
 - Execute TPC-DS queries using Spark operator on Tanzu Kubernetes Cluster(TKG) and incorporating NVIDIA GPUs
 - Validate Spark Performance with Tanzu and NVIDIA GPU leveraging RAPIDS
2. Machine Learning with XGBOOST
 - Machine learning mortgage data using Spark operator with XGBOOST on Tanzu Kubernetes Cluster (TKG) incorporating NVIDIA GPUs

3.5 Building Blocks of the solution:

Function	Components
Server Nodes	Dell R740 with NVIDIA Tesla T4 GPU
Platform	VMware VCF 4 with vSphere 7 VMware TKG 1.2.1 with Kubernetes 1.18
Software	Apache Spark 3.0.1 NVIDIA CUDA 10.2 SPARK RAPIDS 2.12 XGBOOST
Networking	Extreme 10/25 GbE Switches Mellanox 100 GBPS RoCE Switches
Storage	NFS & FC SAN: Infinibox (Infinidat) F6240 Petabyte Scale Storage

Figure 5: HW and Software components of the solutions

4 Use Case 1: TPC-DS with NVIDIA RAPIDS

NVIDIA RAPIDS:

RAPIDS offers a powerful GPU DataFrame based on Apache Arrow data structures. Arrow specifies a standardized, language-independent, columnar memory format, optimized for data locality, to accelerate analytical processing performance on modern CPUs or GPUs. With the GPU DataFrame, batches of column values from multiple records take advantage of modern GPU designs and accelerate reading, queries, and writing.

See <https://developer.nvidia.com/blog/accelerating-apache-spark-3-0-with-gpus-and-rapids/> for more details.

In this use case NVIDIA RAPIDS is run with Apache Spark 3 and validated with a subset of TPC-DS queries. Spark and other components needed for the solution are deployed. Approximately 1TB of data is generated to run TPC-DS against. A subset of TPC-DS queries are then executed with CPU only followed by GPU based acceleration and compared.

Deployment steps:

4.1 Installation of Spark operator for Kubernetes

Spark operator for Kubernetes was installed using the steps in the [Quick Start Guide](#)

- The configuration, `--set enableWebhook=true` was used during installation of the operator in order for volume mounts to work in a spark operator application.
- A helm repository is then added:
 - `helm repo add incubator https://charts.helm.sh/incubator --force-update`
- Then spark-operator is installed. Webhook parameter needs to be set to true for the volume mounts to work. Note that in our case, we had given the operator a name: `sparkoperator-ssgash`
 - `helm install sparkoperator-ssgash incubator/sparkoperator --namespace spark-operator --create-namespace --set sparkJobNamespace=default --set enableWebhook=true`
- Then cluster role binding was created as follows:
 - `kubectl create clusterrolebinding add-on-cluster-admin --clusterrole=cluster-admin --serviceaccount=kube-system:default`

Spark operator official [documentation](#) was used for the deployment in addition to the [Spark Operator user guide](#).

4.2 Install spark history server

Spark History server is used to view the logs of either already finished or currently running Spark applications. The Spark history server where the logs will be stored on a PVC which is NFS based was deployed based on this [configuration guide](#)

4.3 Creation of docker image to run tpcds application related tasks

Created a custom docker image based on `nvidia/cuda:10.2-devel-ubuntu18.04` that incorporates the following versions of dependency software:

- CUDA (10.2)

- UCX (1.8.1)
 - spark v3.0.1
 - CUDF v0.14
 - Spark RAPIDS v2.12
- a GPU resources script that is needed by spark to make it discover and run tasks on GPUs

Apart from these, installed Java 1,8 SDK, Python 2 and 3 in the customer docker image.

Note that this combination of software was arrived at based on the Dockerfile contents mentioned in the following reference document:

<https://docs.mellanox.com/pages/releaseview.action?pageId=25152352>

The Dockerfile that we used for running tpcds on Kubernetes using spark operator is provided in Appendix A.

4.4 Build TPC-DS scala distribution package

Build TPC-DS scala distribution along with the dependencies as specified in <https://github.com/NVIDIA/spark-rapids/blob/branch-0.3/docs/benchmarks.md>

To run TPC/mortgage related applications with spark-rapids, we need this file:

`rapids-4-spark-integration-tests_2.12-0.3.0-SNAPSHOT-jar-with-dependencies.jar`

For building this application jar related to integration_test in spark/rapids, I referred to:

<https://docs.mellanox.com/pages/releaseview.action?pageId=28938181> (slight variation - we used the branch branch-0.3). To be more specific, search for "To download RAPIDS plugin for Apache Spark" to reach the appropriate section.

In our tpcds related applications, we used the jar file that was built with dependencies.

4.5 Install and configure Harbor

Installed and configured Harbor on a local server so that the custom docker images can be stored locally.

Completed harbor install by following [directions](#) from the official site.

A screenshot of the harbor dashboard is shown below.

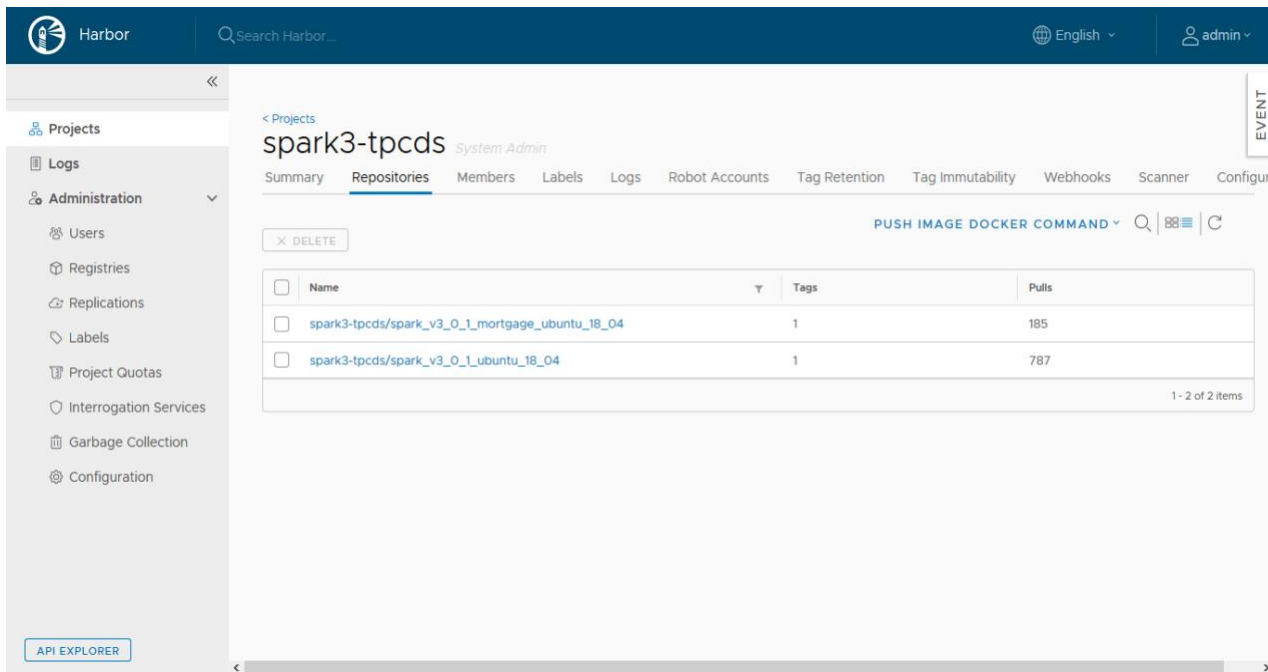


Figure 6: Harbor Repository used as image repository

4.6 TPC-DS Data generation

TPC-DS data was generated as per procedure in the [TPC-DS](#) site.

The data was generated using default options for delimiter (|) for csv file:

```
./dsdgen -DIR /sparknfs/sparkdata/gen_data/1000scale -SCALE 1000
```

The above command generates a 1TB dataset for running TPC-DS queries. This 1TB TPC-DS dataset was converted to parquet format using spark application. Note that you must configure spark-local-dir to have more capacity so that the conversion job doesn't run out of space for storing intermediate files.

Important note: The converted data directories had a '.dat' suffix that needed to be dropped by renaming them in order for the queries to run later.

The yaml file used to convert to parquet format is given in Appendix B.

4.7 Configure Kubernetes worker nodes for running GPU applications

Installed additional software on all kubernetes nodes as well as a kubernetes plugin to enable the cluster to run GPU applications, namely:

- nvidia-docker installation and configuration (in each node)
- nvidia-device-plugin (on the cluster)

Here's a screenshot showing nvidia device plugin in a **Running** state on all the worker nodes:

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
kube-system	nvidia-device-plugin-sc2-2pws	1/1	Running	0	2d21h	192.168.239.77	sc2kubw46
kube-system	nvidia-device-plugin-sc2-8gdw8	1/1	Running	0	2d21h	192.168.124.76	sc2kubw42
kube-system	nvidia-device-plugin-sc2-gkz2q	1/1	Running	0	2d21h	192.168.73.19	sc2kubw43
kube-system	nvidia-device-plugin-sc2-j9gkb	1/1	Running	0	2d21h	192.168.52.9	sc2kubw48
kube-system	nvidia-device-plugin-sc2-m72fx	1/1	Running	0	2d21h	192.168.127.68	sc2kubw45
kube-system	nvidia-device-plugin-sc2-vq22j	1/1	Running	0	2d21h	192.168.166.224	sc2kubw41
kube-system	nvidia-device-plugin-sc2-vwfxh	1/1	Running	0	2d21h	192.168.39.197	sc2kubw47
kube-system	nvidia-device-plugin-sc2-w7dqt	1/1	Running	0	2d21h	192.168.2.135	sc2kubw44

Figure 7: NVIDIA GPU being used across different systems in the worker nodes

4.8 TPC-DS validation testing

In this exercise we ran a select set of TPC-DS queries (chosen based on run times) in both CPU mode and GPU mode. The GPU mode uses the RAPIDS accelerator and Unified Communication X (UCX). UCX provides an optimized communication layer for Message Passing ([MPI](#)), [PGAS/OpenSHMEM](#) libraries and RPC/data-centric applications. UCX utilizes high-speed networks for inter-node communication, and shared memory mechanisms for efficient intra-node communication.

In the CPU mode we used AQE ("adaptive query execution"). This is done by setting spark configuration `"spark.sql.adaptive.enabled": "true"` as shown in [AQE to speed up Spark SQL at runtime](#)

Currently GPU runs cannot be run with AQE ("adaptive query execution") enabled. In both the CPU and GPU cases, we ran a spark application that used one driver and 8 executors.

4.9 TPC-DS Results

TPC-DS queries were run using 1TB TPC-DS data that was converted to parquet format.

The comparative runtimes of these queries are shown in the table below:

TPC-DS Query	GPU Duration	CPU Duration
q98	1.7 min	1.4 min
q73	1.2 min	1.1 min
q68	1.9 min	2.1 min
q64	6.8 min	22 min
q63	1.3 min	1.2 min
q55	1.3 min	1.3 min
q52	1.3 min	1.4 min
q10	1.5 min	1.4 min
q7	1.6 min	1.5 min
q6	1.6 min	6.8 min
q5	3.0 min	4.7 min
q4	11 min	13 min

Figure 8: Runtime comparison of select TPC-DS queries between CPUs and GPUs

The results show that for longer running transactions in many cases that the GPUs accelerate performance. As we can see, GPU makes a significant improvement for the queries q6 and q64 and makes a noticeable improvement for the queries q4 and q5. The remaining queries which have shorter duration, CPU and GPU run with almost similar times.

5 Use Case 2: Model mortgage data using Spark XGBoost running on Kubernetes with Spark Operator

In this use case we use an example mortgage applications to demonstrate the RAPIDS.ai GPU-accelerated XGBoost-Spark project. Further details of the Spark XGBoost examples are described in the following page:

This repo provides docs and example applications that demonstrate the RAPIDS.ai GPU-accelerated XGBoost-Spark project. The scala programs in this reference were used for the solution. The Mortgage data from the link provided on this page are used for the validation.

5.1 Use Case Prerequisites

- Apache Spark 3.0+ (e.g.: Spark 3.0)
- Hardware Requirements
 - NVIDIA Pascal™ GPU architecture or better
 - Multi-node clusters with homogenous GPU configuration
- Software Requirements
 - Ubuntu 16.04/CentOS7
 - CUDA V10.1/10.2)
 - NVIDIA driver compatible with your CUDA
 - NCCL 2.4.7
- [Kubernetes 1.6+ cluster with NVIDIA GPUs](#)
 - See official [Spark on Kubernetes](#) instructions for detailed spark-specific cluster requirements
- kubectl installed and configured in the job submission environment
 - Required for managing jobs and retrieving logs

5.2 Create docker image to run Spark XGBoost application related tasks

Created a custom docker image based on nvidia/cuda:10.2-devel-ubuntu18.04 with XGBoost libraries. This [article](#) was used as a reference for creation of XGBoost image using Docker and was very useful in order to get the XGBoost libraries.

5.3 Data leveraged in use case

An example mortgage application was used to demonstrate the efficacy of XGBOOST leveraging GPUs with Apache Spark and Kubernetes. The Mortgage data was downloaded from [this location](#) and data for the years 2000 and 2001 was used in our deployment. Dataset is derived from [Fannie Mae's Single-Family Loan Performance Data](#).

Fannie Mae provides loan performance data on a portion of its single-family mortgage loans to promote better understanding of the credit performance of Fannie Mae mortgage loans. The population includes two datasets. The Single-Family Fixed Rate Mortgage (primary) dataset contains a subset of Fannie Mae's 30-year and less, fully amortizing, full documentation, single-family, conventional fixed-rate mortgages. The HARP dataset contains approximately one million 30-year fixed rate mortgage loans that are in the primary dataset that were acquired by Fannie Mae from January 1, 2000 through September 30, 2015 and then subsequently refinanced into a fixed rate mortgage through HARP from April 1, 2009 through September 30, 2016.

5.4 Build Scala distribution jars for Mortgage ML

We followed the instructions to build Scala distribution jar file for Mortgage ML programs that use XGBoost libraries as given in the [Maven Build instructions XGBoost scala code for mortgage](#).

5.5 ETL conversion of Mortgage data

Before running ML training on the mortgage data, the data had to be converted using ETL programs. The process described in this [guide](#) was used to convert the raw mortgage data to a format compatible with that of the ML program. Some trial and error was used to get the right set of arguments for the Scala version of the ETL program.

The relative sizes of the data before and after ETL conversion of mortgage data corresponding to the year 2000 are shown. The directory “/sparknfs/sparkdata/mortgage-data/m2000/parque_out/data/” contains the output of ETL program in parquet format as shown in the figure below.

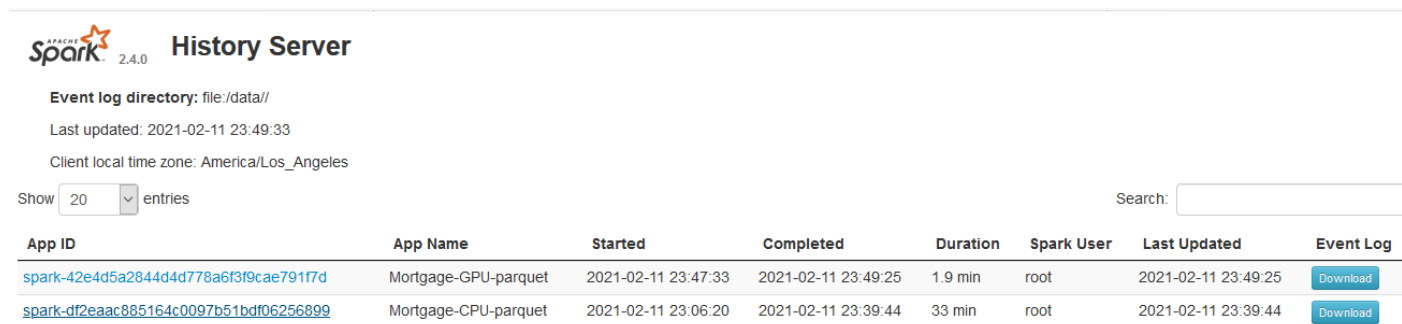
```
root@sc2kubm90:/home/ssgash/shell-scripts# du -sk /sparknfs/sparkdata/mortgage-data/m2000/acq
140096 /sparknfs/sparkdata/mortgage-data/m2000/acq
root@sc2kubm90:/home/ssgash/shell-scripts# du -sk /sparknfs/sparkdata/mortgage-data/m2000/perf
3859776 /sparknfs/sparkdata/mortgage-data/m2000/perf
root@sc2kubm90:/home/ssgash/shell-scripts# du -sk /sparknfs/sparkdata/mortgage-data/m2000/parque_out/data/
320065 /sparknfs/sparkdata/mortgage-data/m2000/parque_out/data/
root@sc2kubm90:/home/ssgash/shell-scripts#
```

Figure 9: ETL Data output in Parquet format

5.6 Mortgage ML training

XGBoost was leveraged to run ML training on 1 year's mortgage data. As the results below in the Spark history server shows the GPU run was much faster than CPU run.

- *CPU took 33 minutes to do training.*
- *GPU completed training in just under 2 minutes.*



The screenshot shows the Apache Spark 2.4.0 History Server interface. It displays a table of training runs with columns for App ID, App Name, Started, Completed, Duration, Spark User, Last Updated, and Event Log. Two runs are visible: 'Mortgage-GPU-parquet' which completed in 1.9 minutes, and 'Mortgage-CPU-parquet' which took 33 minutes. The GPU run is significantly faster.

App ID	App Name	Started	Completed	Duration	Spark User	Last Updated	Event Log
spark-42e4d5a2844d4d778a6f3f9cae791f7d	Mortgage-GPU-parquet	2021-02-11 23:47:33	2021-02-11 23:49:25	1.9 min	root	2021-02-11 23:49:25	Download
spark-df2eac885164c0097b51bdf06256899	Mortgage-CPU-parquet	2021-02-11 23:06:20	2021-02-11 23:39:44	33 min	root	2021-02-11 23:39:44	Download

Figure 10: XGBoost runtime comparison between CPUs and GPUs on Spark History Server

The results clearly show the massive 16X acceleration on GPU training runs with XGBoost versus CPU.

5.7 Mortgage ML test

As is customary in testing, a small portion of the data not used in the training was used as the test data. We created "test" data with one quarter's mortgage data (Q1-2001) using ETL to prepare the data. Then we tested the model created in the training run in the previous step.

The results below shows that the trained model is efficacious with a 98% accuracy on the sample that we used:

```
==> Benchmark: Accuracy for [Mortgage GPU Accuracy parquet stub Unknown Unknown Unknown]:
0.9873203857439782
```

6 Conclusion

We successfully deployed GPU accelerated Apache Spark 3 on VMware Tanzu Kubernetes Grid in this solution. The solution effectively demonstrated that

- VMware Cloud Foundation is a great platform for Apache Spark 3
- VMware support for GPUs and Kubernetes can be effectively used with Apache Spark 3
- TPC-DS with NVIDIA RAPIDS and GPUs were effectively showcased in the solution
- NVIDIA GPUs were used with Kubernetes for Machine Learning with XGBoost

7 Appendix A.

This is the listing of the Dockerfile used to create image to run TPC-DS related spark applications with spark-rapids.

```
# Copyright (c) 2019, NVIDIA CORPORATION. All rights reserved.
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements. See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License. You may obtain a copy of the License at
```

```
# http://www.apache.org/licenses/LICENSE-2.0
```

```
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
# express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```

```
ARG CUDA_VER=10.2
```

```
ARG spark_uid=185
```

```
# Use this base image
```

```
FROM nvidia/cuda:${CUDA_VER}-devel-ubuntu18.04
```

```
# Set MOFED version, OS version and platform
```

```
ENV MOFED_VER 5.0-2.1.8.0
```

```
ENV OS_VER ubuntu18.04
```

```
ENV PLATFORM x86_64
```

```
ENV DEBIAN_FRONTEND=noninteractive
```

```
# Set Unified Communication X {UCX} version, OS, MOFED and CUDA
versions
```

```
ENV UCX_VER v1.8.1
```

```
ENV OS_VER ubuntu18.04
```

```
ENV MOFED mofed5.0
```

```
ENV CUDA 10.2
```

```
# Set Spark, Hadoop, cuDF and RAPIDS versions
```

```
ENV SPARK_VER 3.0.1
```

```
ENV HADOOP_VER 2.7
```

```
ENV CUDF_VER 0.14
```

```
ENV RAPIDS_VER 0.1.0
```

```
ENV RAPIDS 4
```

```
ENV SPARK_RAPIDS spark_2.12
```

```
ENV CUDA_RAPIDS cuda10-2
```

```
# Install dependencies
```

```
RUN apt-get update \
```

```
&& apt install -y git wget apt-utils scala libnuma1 udev libudev1 libcap2
dpkg-reconfigure --frontend noninteractive tzdata
```

```
RUN ln -fs /usr/share/zoneinfo/America/New_York /etc/localtime
```

```
RUN dpkg-reconfigure --frontend noninteractive tzdata
```

```
# Install java dependencies
```

```
-----
RUN apt-get update \
  && apt install -y --no-install-recommends openjdk-8-jdk openjdk-8-jre \
  && rm -rf /var/lib/apt/lists/ && rm -rf /root/.cache && rm -rf /var/cache/apt/*
```

```
ENV JAVA_HOME /usr/lib/jvm/java-1.8.0-openjdk-amd64
```

```
ENV PATH $PATH:/usr/lib/jvm/java-1.8.0-openjdk-
```

```
amd64/jre/bin:/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin
```

```
-----
# Install python 2 and python 3
```

```
# Install numpy and pandas for XGBoost python api tests
```

```
-----
RUN mkdir -p /opt/spark/python
```

```
RUN apt-get update \
```

```
&& apt install -y python python-pip \
```

```
&& apt install -y python3 python3-pip \
```

```
&& pip install --upgrade pip \
```

```
&& pip3 install --upgrade pip \
```

```
&& pip install numpy pandas \
```

```
&& pip3 install numpy pandas \
```

```
# We remove ensurepip since it adds no functionality since pip is
```

```
# installed on the image and it just takes up 1.6MB on the image
```

```
&& rm -r /usr/lib/python*/ensurepip \
```

```
&& pip install --upgrade pip setuptools \
```

```
# You may install with python3 packages by using pip3.6
```

```
# Removed the .cache to save space
```

```
&& rm -rf /var/lib/apt/lists/ && rm -rf /root/.cache && rm -rf /var/cache/apt/*
```

```
-----
# MOFED install
```

```
ENV OFED_FQN MLNX_OFED_LINUX-${MOFED_VER}-${OS_VER}-
```

```
-${PLATFORM}
```

```
RUN wget --quiet http://content.mellanox.com/ofed/MLNX\_OFED-
```

```
\_\${MOFED\_VER}/\${OFED\_FQN}.tgz && \
```

```
tar -xzf ${OFED_FQN}.tgz && \
```

```
${OFED_FQN}/mlnxofedinstall --user-space-only --without-fw-update -q
```

```
RUN cd .. && \
```

```
rm -rf ${MOFED_DIR} && \
```

```
rm -rf *.tgz
```

```
-----
# UCX install
```

```
-----
RUN wget
```

```
https://github.com/openucx/ucx/releases/download/\${UCX\_VER}/ucx-
```

```
\_\${UCX\_VER}-\${OS\_VER}-\${MOFED}-cuda\${CUDA}.deb && \
```

```
dpkg -i ucx-${UCX_VER}-${OS_VER}-${MOFED}-cuda${CUDA}.deb
```

```
-----
# Before building the docker image, first build and make a Spark distribution
following
```

```
# the instructions in http://spark.apache.org/docs/latest/building-spark.html.
```

```
# If this docker file is being used in the context of building your images from a
```

```
Spark
```

```
# distribution, the docker build command should be invoked from the top
```

```
level directory
```

```
# of the Spark distribution. E.g.:
```

```
# docker build -t spark:latest -f kubernetes/dockerfiles/spark/Dockerfile .
```

```
-----
RUN set -ex && \
```

```
ln -s /lib /lib64 && \
```

```
mkdir -p /opt/spark && \
```

```
mkdir -p /opt/spark/jars && \
```

```
mkdir -p /opt/tpch && \
```

```
mkdir -p /opt/spark/examples && \
```

```
mkdir -p /opt/spark/work-dir && \
```

```
mkdir -p /opt/sparkRapidsPlugin && \
```

```
touch /opt/spark/RELEASE && \
```



```

rm /bin/sh && \
ln -sv /bin/bash /bin/sh && \
echo "auth required pam_wheel.so use_uid" >> /etc/pam.d/su && \
chgrp root /etc/passwd && chmod ug+rw /etc/passwd

COPY spark-${SPARK_VER}-bin-hadoop${HADOOP_VER}/jars
/opt/spark/jars
COPY spark-${SPARK_VER}-bin-hadoop${HADOOP_VER}/bin
/opt/spark/bin
COPY spark-${SPARK_VER}-bin-hadoop${HADOOP_VER}/sbin
/opt/spark/sbin
COPY spark-${SPARK_VER}-bin-
hadoop${HADOOP_VER}/kubernetes/dockerfiles/spark/entrypoint.sh /opt/
COPY spark-${SPARK_VER}-bin-hadoop${HADOOP_VER}/examples
/opt/spark/examples
COPY spark-${SPARK_VER}-bin-
hadoop${HADOOP_VER}/kubernetes/tests/opt/spark/tests
COPY spark-${SPARK_VER}-bin-hadoop${HADOOP_VER}/data
/opt/spark/data

#-----
# To run TPC/mortgage related applications with spark-rapids, need this file:
# rapids-4-spark-integration-tests_2.12-0.3.0-SNAPSHOT-jar-with-
dependencies.jar
# This file was created using a docker image that ran the base image,
"nvidia/cuda:10.2-devel-ubuntu18.04"
# where we added all non GPU pieces needed to this as per that Dockerfile
and then did a mvn build
# as given in
https://docs.mellanox.com/pages/releaseview.action?pagelid=28938181
(slight variation -
# we used the branch branch-0.3)
# The following file is the jar file that contains even the dependencies.
COPY jars/rapids-4-spark-integration-tests_2.12-0.3.0-SNAPSHOT-jar-with-
dependencies.jar /opt/sparkRapidsPlugin

#-----
# Download RAPIDS Spark, cuDF Packages and a get GPU resources script
RUN cd /opt/sparkRapidsPlugin && \
wget https://repo1.maven.org/maven2/com/nvidia/rapids-\${RAPIDS\_VER}-
\${SPARK\_RAPIDS}/\${RAPIDS\_VER}/rapids-\${RAPIDS\_VER}-
\${SPARK\_RAPIDS}/\${RAPIDS\_VER}.jar && \

```

```

wget https://repo1.maven.org/maven2/ai/rapids/cudf/\${CUDF\_VER}/cudf-
\${CUDF\_VER}-\${CUDA\_RAPIDS}.jar && \
wget
https://raw.githubusercontent.com/apache/spark/master/examples/src/main/s
cripts/getGpusResources.sh

# Created the following file, getSRIOVResources.sh, as given in
https://docs.mellanox.com/pages/releaseview.action?pagelid=25152352
COPY jars/getSRIOVResources.sh
/opt/sparkRapidsPlugin/getSRIOVResources.sh

COPY spark-${SPARK_VER}-bin-hadoop${HADOOP_VER}/python/pyspark
/opt/spark/python/pyspark
COPY spark-${SPARK_VER}-bin-hadoop${HADOOP_VER}/python/lib
/opt/spark/python/lib

ENV SPARK_HOME /opt/spark
WORKDIR /opt/spark/work-dir
RUN chmod g+w /opt/spark/work-dir

```

```

#-----
# Use tini
#-----

ENV TINI_VERSION v0.18.0
ADD https://github.com/krallin/tini/releases/download/\${TINI\_VERSION}/tini
/usr/bin/tini
RUN chmod +rx /usr/bin/tini
RUN chmod -R 777 /opt/sparkRapidsPlugin

ENTRYPOINT [ "/opt/entrypoint.sh" ]

#-----
# Specify the User that the actual main process will run as
USER ${spark_uid}

#-----

```

8 Appendix B.

This is the yaml used to run tpccs data conversion to parquet format.

```
#
# Copyright 2017 Google LLC
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
# express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication

metadata:
  name: spark-convert
  namespace: default
spec:
  type: Scala
  mode: cluster
  image: "sc2k8cl2.vslab.local/spark3-tpccs/spark_v3_0_1_ubuntu_18_04"
  imagePullPolicy: IfNotPresent
  #
  mainClass: com.nvidia.spark.rapids.tests.tpccs.ConvertFiles
  mainApplicationFile: "local:///gpu_data/mlperf-data/sparkdata/jars/rapids-4-
  spark-integration-tests_2.12-0.3.0-SNAPSHOT-jar-with-dependencies.jar"
  arguments: ["--input", "/sparknfs/sparkdata/gen_data/1000scale", "--output",
  "/sparknfs/sparkdata/gen_data/1000scale_parq", "--output-format", "parquet",
  "--coalesce", "customer_address=1", "--repartition", "web_sales=256",
  "inventory=128"]
  sparkVersion: "3.0.0"
  restartPolicy:
    type: Never
  volumes:
    - name: "sparkcode"
      nfs:
        server: "172.16.35.40"
        path: "/GPU_DB"
        readOnly: false
    - name: "sparkdata"
      nfs:
        server: "172.16.35.60"
        path: "/SPARKNFS01"
        readOnly: false
    - name: "pvc-storage"
      persistentVolumeClaim:
        claimName: nfs
    - name: "spark-local-dir-1"
      hostPath:
        path: "/tmp/spark-local-dir"
  sparkConf:
    # Enable to store the event log
    "spark.eventLog.enabled": "true"
    #Location where to store event log - match the pvc
    "spark.eventLog.dir": "file:/mnt"
  #
  # Restart policies (if I did not specify this, it kept trying executors 500+
  times and exceeded the limits for image-pull).
  restartPolicy:
    type: Never
  driver:
    cores: 1
    coreLimit: "1200m"
    memory: "16384m"
    labels:
```

```
version: 3.0.0
serviceAccount: sparkoperator-ssgash-spark
volumeMounts:
  # name(s) must match the volume name(s) above
  - name: "sparkcode"
    mountPath: "/gpu_data"
  - name: "sparkdata"
    mountPath: "/sparknfs"
  - name: "pvc-storage"
    mountPath: "/mnt"
executor:
  cores: 1
  instances: 6
  memory: "16384m"
  labels:
    version: 3.0.0
  volumeMounts:
    - name: "sparkcode"
      mountPath: "/gpu_data"
    - name: "sparkdata"
      mountPath: "/sparknfs"
    - name: "pvc-storage"
      mountPath: "/mnt"
    - name: "spark-local-dir-1"
      mountPath: "/tmp/spark-local-dir"
```

9 Appendix C.

This is the template yaml file used to run tpcds queries on spark operator using CPU:

```
#
# Copyright 2017 Google LLC
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication

metadata:
  name: spark-tpcds-%CPU_OR_GPU%-%QUERY_NAME%-run
  namespace: default
spec:
  type: Scala
  mode: cluster
  image: "sc2k8cl2.vslab.local/spark3-tpcds/spark_v3_0_1_ubuntu_18_04"
  imagePullPolicy: Always
  #
  mainClass: com.nvidia.spark.rapids.tests.BenchmarkRunner
  mainApplicationFile: "local:///opt/sparkRapidsPlugin/rapids-4-spark-integration-tests_2.12-0.3.0-SNAPSHOT-jar-with-dependencies.jar"
  # Other spark-submit arguments for the specific application
  # SCALE = 1000
  arguments: ["--benchmark", "tpcds", "--query", "%QUERY_NAME%", "--input", "/sparknfs/sparkdata/gen_data/1000scale_parq", "--input-format", "parquet", "--output", "/sparknfs/sparkdata/gen_data/1000scale_parq/tpcds-output/tpcds-%QUERY_NAME%-%UNIQUE_SUFFIX%-%CPU_OR_GPU%", "--output-format", "parquet", "--summary-file-prefix", "/sparknfs/sparkdata/gen_data/1000scale_parq/tpcds-output/tpcds-%QUERY_NAME%-%UNIQUE_SUFFIX%-%CPU_OR_GPU%", "--iterations", "1"]
  #
  sparkVersion: "3.0.0"
  restartPolicy:
    type: Never
  # volumes:
  volumes:
  - name: "sparkcode"
    nfs:
      server: "172.16.35.40"
      path: "/GPU_DB"
      readOnly: false
  - name: "sparkdata"
    nfs:
      server: "172.16.35.60"
      path: "/SPARKNFS01"
      readOnly: false
  - name: "pvc-storage"
    persistentVolumeClaim:
      claimName: nfs
  - name: "spark-local-dir-1"
    hostPath:
      path: "/tmp/spark-local-dir"
  sparkConf:
    # Adaptive QueryExecution - supported in spark 3.x and above - turning it
    on.
    "spark.sql.adaptive.enabled": "true"
    "spark.sql.broadcastTimeout": "600"
    # Enable to store the event log
    "spark.eventLog.enabled": "true"
    #Location where to store event log - match the pvc
    "spark.eventLog.dir": "file:/mnt"
    # Logs the effective SparkConf as INFO when a SparkContext is started
    "spark.logConf": "true"
```

Driver and executor configs

```
driver:
  cores: 1
  coreLimit: "1200m"
  memory: "12G"
  labels:
    version: 3.0.0
  serviceAccount: sparkoperator-ssgash-spark
  volumeMounts:
    # name(s) must match the volume name(s) above
    - name: "sparkcode"
      mountPath: "/gpu_data"
    - name: "sparkdata"
      mountPath: "/sparknfs"
    - name: "pvc-storage"
      mountPath: "/mnt"
```

```
executor:
  cores: 1
  instances: 8
  memory: "16G"
  labels:
    version: 3.0.0
  volumeMounts:
    # name(s) must match the volume name(s) above
    - name: "sparkcode"
      mountPath: "/gpu_data"
    - name: "sparkdata"
      mountPath: "/sparknfs"
    - name: "pvc-storage"
      mountPath: "/mnt"
    - name: "spark-local-dir-1"
      mountPath: "/tmp/spark-local-dir"
```

10 Appendix D.

This is the template yaml file used to run tpcds queries on spark operator using GPU:

```
#
# Copyright 2017 Google LLC
#
# Licensed under the Apache License, Version 2.0 (the "License");
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication

metadata:
  name: spark-tpcds-%CPU_OR_GPU%-%QUERY_NAME%-run
  namespace: default
spec:
  type: Scala
  mode: cluster
  #Now using local harbor as docker repo - with a newly built image.
  image: "sc2k8cl2.vslab.local/spark3-tpcds/spark_v3_0_1_ubuntu_18_04"
  imagePullPolicy: Always
  #
  mainClass: com.nvidia.spark.rapids.tests.BenchmarkRunner
  mainApplicationFile: "local:///opt/sparkRapidsPlugin/rapids-4-spark-integration-tests_2.12-0.3.0-SNAPSHOT-jar-with-dependencies.jar"
  # Other spark-submit arguments for the specific application
  # Running just one query: %QUERY_NAME%
  # SCALE = 1000
  arguments: ["--benchmark", "tpcds", "--query", "%QUERY_NAME%", "--input", "/sparknfs/sparkdata/gen_data/1000scale_parq", "--input-format", "parquet", "--output", "/sparknfs/sparkdata/gen_data/1000scale_parq/tpcds-output/tpcds-%QUERY_NAME%-%UNIQUE_SUFFIX%-%CPU_OR_GPU%-newnodes", "--output-format", "parquet", "--summary-file-prefix", "/sparknfs/sparkdata/gen_data/1000scale_parq/tpcds-output/tpcds-%QUERY_NAME%-%UNIQUE_SUFFIX%-%CPU_OR_GPU%-newnodes", "--iterations", "1"]
  #
  sparkVersion: "3.0.0"
  restartPolicy:
    type: Never
  # volumes:
  volumes:
  - name: "sparkcode"
    nfs:
      server: "172.16.35.40"
      path: "/GPU_DB"
      readOnly: false
  - name: "sparkdata"
    nfs:
      server: "172.16.35.60"
      path: "/SPARKNFS01"
      readOnly: false
  - name: "pvc-storage"
    persistentVolumeClaim:
      claimName: nfs
  - name: "spark-local-dir-1"
    hostPath:
      path: "/tmp/spark-local-dir"
  sparkConf:
    "spark.sql.broadcastTimeout": "600"

  # Enable to store the event log
  "spark.eventLog.enabled": "true"

  #Location where to store event log - match the pvc
  "spark.eventLog.dir": "file:/mnt"

  # Logs the effective SparkConf as INFO when a SparkContext is started
  "spark.logConf": "true"

  # Following configs are from TCP with GPU/RAPIDS and without UCX:
  # Pinned memory refers to memory pages that the OS will keep in system RAM and will not relocate or swap to disk
  "spark.rapids.memory.pinnedPool.size": "4G"
```

```
"spark.rapids.memory.gpu.pooling.enabled": "true"

"spark.rapids.memory.gpu.allocFraction": "0.5"
"spark.rapids.memory.gpu.maxAllocFraction": "0.70"
"spark.rapids.memory.gpu.debug": "STDOUT"

"spark.task.resource.gpu.amount": "0.25"
"spark.rapids.sql.enabled": "true"

"spark.rapids.sql.concurrentGpuTasks": "1"

"spark.sql.files.maxPartitionBytes": "512m"

"spark.sql.shuffle.partitions": "200"

# UCX config
"spark.shuffle.manager": "com.nvidia.spark.RapidsShuffleManager"
"spark.rapids.shuffle.transport.enabled": "true"
"spark.executorEnv.UCX_TLS": "cuda_copy,cuda_ipc,tcp"
"spark.executorEnv.UCX_NET_DEVICES": "eth0"

# Other spark configs
"spark.locality.wait": "0s"
"spark.driver.extraClassPath": "/opt/sparkRapidsPlugin/**"
"spark.plugins": "com.nvidia.spark.SQLPlugin"

"spark.executor.extraClassPath":
"/opt/sparkRapidsPlugin/*:/usr/lib:/data/jar/**

"spark.executor.resource.gpu.amount": "1"
"spark.executor.resource.gpu.discoveryScript":
"/opt/sparkRapidsPlugin/getGpusResources.sh"
"spark.executor.resource.gpu.vendor": "nvidia.com"

"spark.executorEnv.LD_LIBRARY_PATH": "/usr/local/cuda/lib64"
"spark.executorEnv.CUDA_HOME": "/usr/local/cuda"

# Driver and executor configs
driver:
  cores: 1
  coreLimit: "1200m"
  memory: "12G"
  labels:
    version: 3.0.0
  serviceAccount: sparkoperator-ssgash-spark
# volumeMounts:
volumeMounts:
  # name(s) must match the volume name(s) above
  - name: "sparkcode"
    mountPath: "/gpu_data"
  - name: "sparkdata"
    mountPath: "/sparknfs"
  - name: "pvc-storage"
    mountPath: "/mnt"

executor:
  cores: 1
  instances: 8
  memory: "16G"
  labels:
    version: 3.0.0
# volumeMounts:
volumeMounts:
  # name(s) must match the volume name(s) above
  - name: "sparkcode"
    mountPath: "/gpu_data"
  - name: "sparkdata"
    mountPath: "/sparknfs"
  - name: "pvc-storage"
    mountPath: "/mnt"
  - name: "spark-local-dir-1"
    mountPath: "/tmp/spark-local-dir"
```

11 Appendix E.

This is the listing of the Dockerfile used to create image to run Mortgage ML related spark applications with XGBoost.

```
# Copyright (c) 2019, NVIDIA CORPORATION. All rights reserved.
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements. See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License. You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
# express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
#-----
ARG CUDA_VER=10.2
ARG spark_uid=185
#-----

# Use this base image
FROM nvidia/cuda:${CUDA_VER}-devel-ubuntu18.04

#-----
# Set MOFED version, OS version and platform
#-----
ENV MOFED_VER 5.0-2.1.8.0
ENV OS_VER ubuntu18.04
ENV PLATFORM x86_64
ENV DEBIAN_FRONTEND=noninteractive

#-----
# Set Unified Communication X {UCX} version, OS, MOFED and CUDA
# versions
#-----
ENV UCX_VER v1.8.1
ENV OS_VER ubuntu18.04
ENV MOFED mofed5.0
ENV CUDA 10.2

#-----
# Set Spark, Hadoop, cuDF and RAPIDS versions
#-----
ENV SPARK_VER 3.0.1
ENV HADOOP_VER 2.7
ENV CUDF_VER 0.14
ENV RAPIDS_VER 0.1.0
ENV RAPIDS 4
ENV SPARK_RAPIDS spark_2.12
ENV CUDA_RAPIDS cuda10-2

#-----
# Install dependencies
#-----
RUN apt-get update \
  && apt install -y git wget apt-utils scala libnuma1 udev libudev1 libcap2
  dpatch libnl-3-200 gfortran automake lsof ethtool chrpath libmnl0 pkg-config
  m4 libnl-route-3-200 autoconf debhelper swig bison libltdl-dev kmod tcl libnl-
  route-3-dev pciutils tk autotools-dev flex libnl-3-dev graphviz libgfortran4
  iproute2 iputils-ping \
  && rm -rf /var/lib/apt/lists/ && rm -rf /root/.cache && rm -rf /var/cache/apt/*

RUN ln -fs /usr/share/zoneinfo/America/New_York /etc/localtime
RUN dpkg-reconfigure --frontend noninteractive tzdata

#-----
```

```
# Install java dependencies
#-----
RUN apt-get update \
  && apt install -y --no-install-recommends openjdk-8-jdk openjdk-8-jre \
  && rm -rf /var/lib/apt/lists/ && rm -rf /root/.cache && rm -rf /var/cache/apt/*

ENV JAVA_HOME /usr/lib/jvm/java-1.8.0-openjdk-amd64
ENV PATH $PATH:/usr/lib/jvm/java-1.8.0-openjdk-
amd64/jre/bin:/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin

#-----
# Install python 2 and python 3
# Install numpy and pandas for XGBoost python api tests
#-----
RUN mkdir -p /opt/spark/python
RUN apt-get update \
  && apt install -y --no-install-recommends python python-pip libgomp1 \
  && apt install -y --no-install-recommends python3 python3-pip \
  && pip install --upgrade pip \
  && pip3 install --upgrade pip \
  && pip install numpy pandas \
  && pip3 install numpy pandas \
# We remove ensurepip since it adds no functionality since pip is
# installed on the image and it just takes up 1.6MB on the image
  && rm -r /usr/lib/python*/ensurepip \
  && pip install --upgrade pip setuptools \
# You may install with python3 packages by using pip3.6
# Removed the .cache to save space
  && rm -rf /var/lib/apt/lists/ && rm -rf /root/.cache && rm -rf /var/cache/apt/*

RUN apt-get update \
  && apt-get install -y --no-install-recommends apt-utils \
# Removed the .cache to save space
  && rm -rf /var/lib/apt/lists/ && rm -rf /root/.cache && rm -rf /var/cache/apt/*

#-----
# MOFED install
#-----
ENV OFED_FQN MLNX_OFED_LINUX-${MOFED_VER}-${OS_VER}-
${PLATFORM}
RUN wget --quiet http://content.mellanox.com/ofed/MLNX_OFED-
${MOFED_VER}/${OFED_FQN}.tgz && \
tar -xf ${OFED_FQN}.tgz && \
${OFED_FQN}/mlnxofedinstall --user-space-only --without-fw-update -q
RUN cd .. && \
rm -rf ${MOFED_DIR} && \
rm -rf *.tgz

#-----
# UCX install
#-----
RUN wget
https://github.com/openucx/ucx/releases/download/${UCX_VER}/ucx-
${UCX_VER}-${OS_VER}-${MOFED}-cuda${CUDA}.deb && \
dpkg -i ucx-${UCX_VER}-${OS_VER}-${MOFED}-cuda${CUDA}.deb

#-----
# Copy spark3 directories
#-----
# Before building the docker image, first build and make a Spark distribution
following
# the instructions in http://spark.apache.org/docs/latest/building-spark.html.
# If this docker file is being used in the context of building your images from a
Spark
# distribution, the docker build command should be invoked from the top
level directory
# of the Spark distribution. E.g.:
# docker build -t spark:latest -f kubernetes/dockerfiles/spark/Dockerfile .

RUN set -ex && \
ln -s /lib /lib64 && \
```

```
mkdir -p /opt/spark && \
mkdir -p /opt/spark/jars && \
mkdir -p /opt/spark/data && \
mkdir -p /opt/spark/examples && \
mkdir -p /opt/spark/work-dir && \
mkdir -p /opt/sparkRapidsPlugin && \
touch /opt/spark/RELEASE && \
rm /bin/sh && \
ln -sv /bin/bash /bin/sh && \
echo "auth required pam_wheel.so use_uid" >> /etc/pam.d/su && \
chgrp root /etc/passwd && chmod ug+rw /etc/passwd
```

```
COPY spark-${SPARK_VER}-bin-hadoop${HADOOP_VER}/jars
/opt/spark/jars
COPY spark-${SPARK_VER}-bin-hadoop${HADOOP_VER}/bin
/opt/spark/bin
COPY spark-${SPARK_VER}-bin-hadoop${HADOOP_VER}/sbin
/opt/spark/sbin
COPY spark-${SPARK_VER}-bin-
hadoop${HADOOP_VER}/kubernetes/dockerfiles/spark/entrypoint.sh /opt/
COPY spark-${SPARK_VER}-bin-hadoop${HADOOP_VER}/examples
/opt/spark/examples
COPY spark-${SPARK_VER}-bin-
hadoop${HADOOP_VER}/kubernetes/tests /opt/spark/tests
COPY spark-${SPARK_VER}-bin-hadoop${HADOOP_VER}/data
/opt/spark/data
```

```
ENV SPARK_HOME /opt/spark
```

```
#-----
# Install XGBoost
#-----
```

```
RUN pushd $SPARK_HOME/jars \
&& wget
https://repo1.maven.org/maven2/com/nvidia/XGBoost4j_3.0/1.0.0-
0.1.0/XGBoost4j_3.0-1.0.0-0.1.0.jar \
&& wget https://repo1.maven.org/maven2/com/nvidia/XGBoost4j-
spark_3.0/1.0.0-0.1.0/XGBoost4j-spark_3.0-1.0.0-0.1.0.jar \
&& popd
```

```
#-----
# Sample XGBoost scala jar
# Build instructions : https://github.com/NVIDIA/spark-XGBoost-
examples/blob/spark-3/getting-started-guides/building-sample-apps/scala.md
# Cloned into root@sc2k8cl3:/gpu_data/mlperf-data/sparkdata/ and followed
the instructions.
# With dep: Actual location: /gpu_data/mlperf-data/sparkdata/spark-
XGBoost-examples/examples/apps/scala/target/sample_XGBoost_apps-
0.2.2-jar-with-dependencies.jar
# Without dep: Actual location: /gpu_data/mlperf-data/sparkdata/spark-
XGBoost-examples/examples/apps/scala/target/sample_XGBoost_apps-
0.2.2.jar
# copied to root@sc2k8cl3:/gpu_data/mlperf-data/sparkdata/docker-
builder/jars
# We are using the jar without dependencies.
COPY jars/sample_XGBoost_apps-0.2.2.jar /opt/sparkRapidsPlugin
```

```
#-----
# Download RAPIDS Spark, cuDF Packages and a get GPU resources script
#-----
```

```
RUN cd /opt/sparkRapidsPlugin && \
wget https://repo1.maven.org/maven2/com/nvidia/rapids-${RAPIDS}-
${SPARK_RAPIDS}/${RAPIDS_VER}/rapids-${RAPIDS}-
${SPARK_RAPIDS}-${RAPIDS_VER}.jar && \
wget https://repo1.maven.org/maven2/ai/rapids/cudf/${CUDF_VER}/cudf-
${CUDF_VER}-${CUDA_RAPIDS}.jar && \
wget
https://raw.githubusercontent.com/apache/spark/master/examples/src/main/s
cripts/getGpusResources.sh
```

```
# Created the following file, getSRIOVResources.sh, as given in
https://docs.mellanox.com/pages/releaseview.action?pagelid=25152352
COPY jars/getSRIOVResources.sh
/opt/sparkRapidsPlugin/getSRIOVResources.sh
```

```
COPY spark-${SPARK_VER}-bin-hadoop${HADOOP_VER}/python/pyspark
/opt/spark/python/pyspark
COPY spark-${SPARK_VER}-bin-hadoop${HADOOP_VER}/python/lib
/opt/spark/python/lib
```

```
ENV SPARK_HOME /opt/spark
WORKDIR /opt/spark/work-dir
RUN chmod g+w /opt/spark/work-dir
```

```
#-----
# Use tini
#-----
```

```
ENV TINI_VERSION v0.18.0
ADD https://github.com/krallin/tini/releases/download/${TINI_VERSION}/tini
/usr/bin/tini
RUN chmod +rx /usr/bin/tini
RUN chmod -R 777 /opt/sparkRapidsPlugin
```

```
ENTRYPOINT [ "/opt/entrypoint.sh" ]
```

```
# Specify the User that the actual main process will run as
USER ${spark_uid}
#-----
```

12 Appendix F.

This is the yaml used to convert mortgage data to parquet format.

```
#
# Copyright 2017 Google LLC
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
# express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication

metadata:
  name: spark-convert-mort
  namespace: default
spec:
  type: Scala
  mode: cluster
  image: "sc2k8cl2.vslab.local/spark3-tpcds/spark_v3_0_1_mortgage_ubuntu_18_04"
  imagePullPolicy: Always
  #
  mainClass: com.nvidia.spark.examples.mortgage.ETLMain
  mainApplicationFile:
"local:///opt/sparkRapidsPlugin/sample_XGBoost_apps-0.2.2.jar"
  #
  # For mortgage data conversion, there are just 3 arguments:
  ETLArgs(perfPath: String, acqPath: String, output: String)
  #
  # ETL to convert q1m2000 data so that it can be used to "train" and create
  the model
  #arguments: ["-format=csv", "-
  dataPath=perf::/sparknfs/sparkdata/mortgage-data/q1m2000/perf", "-
  dataPath=acq::/sparknfs/sparkdata/mortgage-data/q1m2000/acq", "-
  dataPath=out::/sparknfs/sparkdata/mortgage-data/q1m2000/pout"]
  #
  # ETL to convert q1m2001 data so that it can be used to "test" the model
  created by m2000
  arguments: ["-format=csv", "-dataPath=perf::/sparknfs/sparkdata/mortgage-
  data/q1m2001/perf", "-dataPath=acq::/sparknfs/sparkdata/mortgage-
  data/q1m2001/acq", "-dataPath=out::/sparknfs/sparkdata/mortgage-
  data/q1m2001/pout"]
  sparkVersion: "3.0.1"
  restartPolicy:
    type: Never
  volumes:
  - name: "sparkcode"
    nfs:
      server: "172.16.35.40"
      path: "/GPU_DB"
      readOnly: false
  - name: "sparkdata"
    nfs:
      server: "172.16.35.60"
      path: "/SPARKNFS01"
      readOnly: false
  - name: "pvc-storage"
    persistentVolumeClaim:
      claimName: nfs
  - name: "spark-local-dir-1"
    hostPath:
      path: "/tmp/spark-local-dir"
  sparkConf:
```

```
# Enable to store the event log
"spark.eventLog.enabled": "true"
#Location where to store event log - match the pvc
"spark.eventLog.dir": "file:/mnt"
#
# Just made it -1 to make conversion job run
"spark.sql.broadcastTimeout": "-1"
"spark.driver.extraClassPath": "/opt/sparkRapidsPlugin/*"
"spark.executor.extraClassPath":
"/opt/sparkRapidsPlugin*/usr/lib/./data/jar/*"
restartPolicy:
  type: Never
driver:
  cores: 1
  memory: "16G"
  labels:
    version: 3.0.0
  serviceAccount: sparkoperator-ssgash-spark
  volumeMounts:
    # name(s) must match the volume name(s) above
    - name: "sparkcode"
      mountPath: "/gpu_data"
    - name: "sparkdata"
      mountPath: "/sparknfs"
    - name: "pvc-storage"
      mountPath: "/mnt"
  executor:
    cores: 1
    instances: 6
    memory: "24G"
    labels:
      version: 3.0.0
    volumeMounts:
      # name(s) must match the volume name(s) above
      - name: "sparkcode"
        mountPath: "/gpu_data"
      - name: "sparkdata"
        mountPath: "/sparknfs"
      - name: "pvc-storage"
        mountPath: "/mnt"
      - name: "spark-local-dir-1"
        mountPath: "/tmp/spark-local-dir"
```

13 Appendix G.

This is the yaml used to train mortgage data (which has been converted to parquet format).

```
#
# Copyright 2017 Google LLC
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
# express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication

metadata:
  name: spark-train-mort-gpu
  namespace: default
spec:
  dynamicAllocation:
    enabled: true
    initialExecutors: 1
    minExecutors: 1
    maxExecutors: 10
  type: Scala
  mode: cluster
  image: "sc2k8cl2.vslab.local/spark3-
tpcds/spark_v3_0_1_mortgage_ubuntu_18_04"
  imagePullPolicy: Always
  #
  mainClass: com.nvidia.spark.examples.mortgage.GPUMain
  mainApplicationFile:
"/local:///opt/sparkRapidsPlugin/sample_XGBoost_apps-0.2.2.jar"
  #
  # Training with Q1-Year2000 mortgage data - which has been ETL'ed
  arguments: ["-format=parquet", "-
dataPath=train::/sparknfs/sparkdata/mortgage-data/q1m2000/pout/data", "-
mode=train", "-modelPath=/sparknfs/sparkdata/mortgage-
data/q1m2000/model", "-treeMethod=hist"]
  #
  # Testing the model with Q1-Year2001 mortgage data - which has been
  ETL'ed
  # arguments: ["-format=parquet", "-
dataPath=trans::/sparknfs/sparkdata/mortgage-data/q1m2000/pout/data", "-
mode=transform", "-modelPath=/sparknfs/sparkdata/mortgage-
data/m2000/model", "-treeMethod=gpu_hist", "-verbosity=3"]
  #
  sparkVersion: "3.0.1"
  restartPolicy:
    type: Never
  volumes:
    - name: "sparkcode"
      nfs:
        server: "172.16.35.40"
        path: "/GPU_DB"
        readOnly: false
    - name: "sparkdata"
      nfs:
        server: "172.16.35.60"
        path: "/SPARKNFS01"
        readOnly: false
    - name: "pvc-storage"
      persistentVolumeClaim:
        claimName: nfs
```

```
- name: "spark-local-dir-1"
  persistentVolumeClaim:
    claimName: pv-claim-demo
sparkConf:
  # Enable to store the event log
  "spark.eventLog.enabled": "true"
  #Location where to store event log - match the pvc
  "spark.eventLog.dir": "file:/mnt"
  #
  # Just made it -1 to make conversion job run
  "spark.sql.broadcastTimeout": "-1"
  #
  "spark.driver.extraClassPath": "/opt/sparkRapidsPlugin/**"
  "spark.executor.extraClassPath":
"/opt/sparkRapidsPlugin*/usr/lib*/data/jar/**"
  #
  # GPU related config
  "spark.plugins": "com.nvidia.spark.SQLPlugin"
  "spark.task.resource.gpu.amount": "0.25"
  "spark.rapids.sql.concurrentGpuTasks": "1"
  "spark.executor.resource.gpu.amount": "1"
  "spark.executor.resource.gpu.discoveryScript":
"/opt/sparkRapidsPlugin/getGpusResources.sh"
  "spark.executor.resource.gpu.vendor": "nvidia.com"
  "spark.rapids.memory.gpu.pooling.enabled": "false"

  "spark.executorEnv.LD_LIBRARY_PATH": "/usr/local/cuda/lib64"
  "spark.executorEnv.CUDA_HOME": "/usr/local/cuda"
  #
restartPolicy:
  type: Never
driver:
  cores: 1
  memory: "16G"
  labels:
    version: 3.0.0
  serviceAccount: sparkoperator-ssgash-spark
volumeMounts:
  # name(s) must match the volume name(s) above
  - name: "sparkcode"
    mountPath: "/gpu_data"
  - name: "sparkdata"
    mountPath: "/sparknfs"
  - name: "pvc-storage"
    mountPath: "/mnt"
executor:
  cores: 1
  instances: 1
  memory: "24G"
  labels:
    version: 3.0.0
  volumeMounts:
    # name(s) must match the volume name(s) above
    - name: "sparkcode"
      mountPath: "/gpu_data"
    - name: "sparkdata"
      mountPath: "/sparknfs"
    - name: "pvc-storage"
      mountPath: "/mnt"
    - name: "spark-local-dir-1"
      mountPath: "/tmp/spark-local-dir"
```


14 Appendix H.

This is the yaml used to test mortgage data model (using test data which has been converted to parquet format).

```
#
# Copyright 2017 Google LLC
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
# express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication

metadata:
  name: spark-test-mort-gpu
  namespace: default
spec:
  dynamicAllocation:
    enabled: true
    initialExecutors: 1
    minExecutors: 1
    maxExecutors: 10
  type: Scala
  mode: cluster
  image: "sc2k8c12.vslab.local/spark3-
tpcds/spark_v3_0_1_mortgage_ubuntu_18_04"
  imagePullPolicy: Always
  #
  mainClass: com.nvidia.spark.examples.mortgage.GPUMain
  mainApplicationFile:
"local:///opt/sparkRapidsPlugin/sample_XGBoost_apps-0.2.2.jar"
  #
  # Testing the model with Q1-Year2001 mortgage data - which has been
  ETL'ed
  arguments: ["-format=parquet", "-
dataPath=trans::/sparknfs/sparkdata/mortgage-data/q1m2000/pout/data", "-
mode=transform", "-modelPath=/sparknfs/sparkdata/mortgage-
data/m2000/model", "-treeMethod=gpu_hist", "-verbosity=3"]
  #
  sparkVersion: "3.0.1"
  restartPolicy:
    type: Never
  volumes:
  - name: "sparkcode"
    nfs:
      server: "172.16.35.40"
      path: "/GPU_DB"
      readOnly: false
  - name: "sparkdata"
    nfs:
      server: "172.16.35.60"
      path: "/SPARKNFS01"
      readOnly: false
  - name: "pvc-storage"
    persistentVolumeClaim:
      claimName: nfs
  - name: "spark-local-dir-1"
    persistentVolumeClaim:
      claimName: pv-claim-demo
  sparkConf:
    # Enable to store the event log
    "spark.eventLog.enabled": "true"
    #Location where to store event log - match the pvc
```

```
"spark.eventLog.dir": "file:/mnt"
#
# Just made it -1 to make conversion job run
"spark.sql.broadcastTimeout": "-1"
#
"spark.driver.extraClassPath": "/opt/sparkRapidsPlugin/**"
"spark.executor.extraClassPath":
"/opt/sparkRapidsPlugin/**:/usr/lib:/data/jar/**"
#
# GPU related config
"spark.plugins": "com.nvidia.spark.SQLPlugin"
"spark.task.resource.gpu.amount": "0.25"
"spark.rapids.sql.concurrentGpuTasks": "1"
"spark.executor.resource.gpu.amount": "1"
"spark.executor.resource.gpu.discoveryScript":
"/opt/sparkRapidsPlugin/getGpusResources.sh"
"spark.executor.resource.gpu.vendor": "nvidia.com"
"spark.rapids.memory.gpu.pooling.enabled": "false"

"spark.executorEnv.LD_LIBRARY_PATH": "/usr/local/cuda/lib64"
"spark.executorEnv.CUDA_HOME": "/usr/local/cuda"
#
restartPolicy:
  type: Never
driver:
  cores: 1
  memory: "16G"
  labels:
    version: 3.0.0
  serviceAccount: sparkoperator-ssgash-spark
  volumeMounts:
    # name(s) must match the volume name(s) above
    - name: "sparkcode"
      mountPath: "/gpu_data"
    - name: "sparkdata"
      mountPath: "/sparknfs"
    - name: "pvc-storage"
      mountPath: "/mnt"
  executor:
    cores: 1
    instances: 1
    memory: "24G"
    labels:
      version: 3.0.0
    volumeMounts:
      # name(s) must match the volume name(s) above
      - name: "sparkcode"
        mountPath: "/gpu_data"
      - name: "sparkdata"
        mountPath: "/sparknfs"
      - name: "pvc-storage"
        mountPath: "/mnt"
      - name: "spark-local-dir-1"
        mountPath: "/tmp/spark-local-dir"
```