Abstract

In this project we tested Dell EMC PowerEdge Servers with end to end (E2E) workflow using New York City - Taxi notebook to accelerate large scale workloads in scale-up and scale-out solutions using RAPID library from NVIDIA™ and DASK-CUDA for parallel computing with XGBoost.

February 2020
Revisions

<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/4/2020</td>
<td>Initial release</td>
</tr>
</tbody>
</table>

Authors

This paper was produced by the following:

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vilmara Sanchez</td>
<td>Dell EMC, Advanced Engineering</td>
</tr>
<tr>
<td>Bhavesh Patel</td>
<td>Dell EMC, Advanced Engineering</td>
</tr>
</tbody>
</table>

Acknowledgements

This paper was supported by the following:

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Josh Anderson</td>
<td>Dell EMC</td>
</tr>
<tr>
<td>Robert Crovella</td>
<td>NVIDIA</td>
</tr>
<tr>
<td>NVIDIA Account Team</td>
<td>NVIDIA</td>
</tr>
<tr>
<td>NVIDIA Developer Forum</td>
<td>NVIDIA</td>
</tr>
<tr>
<td>NVIDIA RAPIDS Dev Team</td>
<td>NVIDIA</td>
</tr>
<tr>
<td>Dask Development Team</td>
<td>Library for dynamic task scheduling <a href="https://dask.org">https://dask.org</a></td>
</tr>
</tbody>
</table>

The information in this publication is provided “as is.” Dell Inc. makes no representations or warranties of any kind with respect to the information in this publication, and specifically disclaims implied warranties of merchantability or fitness for a particular purpose.

Use, copying, and distribution of any software described in this publication requires an applicable software license.

© February, 2020 Dell Inc. or its subsidiaries. All Rights Reserved. Dell, EMC, Dell EMC and other trademarks are trademarks of Dell Inc. or its subsidiaries. Other trademarks may be trademarks of their respective owners.

Dell believes the information in this document is accurate as of its publication date. The information is subject to change without notice.
Table of Contents

1 RAPIDS Overview ........................................................................................................................................... 5
  1.1 XGBoost ....................................................................................................................................................... 6
  1.2 Dask for Parallel Computing ............................................................................................................................ 7
  1.3 Why RAPIDS, Dask and XGBoost? .................................................................................................................... 8
  1.4 New York City (NYC) – Taxi Dataset .................................................................................................................. 8
  1.5 E2E NYC-Taxi Notebook ..................................................................................................................................... 9
  1.6 RAPIDS Memory Manager (RMM) .................................................................................................................... 9
2 System Configuration .......................................................................................................................................... 10
3 Results on Single Node ....................................................................................................................................... 11
4 Results on Multi Node on C4140-M ....................................................................................................................... 13
5 Results with System Profile in “Performance” Mode .............................................................................................. 15
6 Conclusion and Future Work .................................................................................................................................... 17
A Dell EMC PowerEdge Server Specifications ......................................................................................................... 18
B Terminology .......................................................................................................................................................... 19
C Example; GPU Activity with C4140-M in Multi Node Mode ..................................................................................... 20
D Dask Diagnostic Dashboard ..................................................................................................................................... 22
E NVDashboard – Nvidia GPU Dashboard .................................................................................................................. 25
F Environment set up .................................................................................................................................................. 28
G Notebook NYC-Taxi Set Up ..................................................................................................................................... 29
H RAPIDS Multi Node Set Up ..................................................................................................................................... 30
I Bio8 Settings to Boost Performance ............................................................................................................................ 31
J Common Errors ....................................................................................................................................................... 33
K Technical Resources .................................................................................................................................................. 34
  K.1 Related Resources ............................................................................................................................................... 34
Executive Summary

Traditional Machine learning workflows often involve iterative and lengthy steps in data preparation, model training, validating results and tuning models before the final solution can be deployed for production. This cycle can consume a lot of resources, negatively impacting the productivity of the developer’s team toward business transformation. In order to accelerate this, NVIDIA released the Accelerated Data Science pipeline with RAPIDS. It’s a complete ecosystem that integrates multiple Python libraries with CUDA at the core level and built on CUDA-X AI libraries and other key open-source projects including Apache Arrow. This ecosystem provides GPU-accelerated software for data science workflows that maximizes productivity, performance and ROI, at the lowest infrastructure total cost of ownership (TCO).

In this paper we tested the NYC-Taxi sample notebook (included in the NGC RAPIDS container) and the NYC Taxi dataset [1] (available from a public Google Cloud Storage bucket) on Dell EMC PowerEdge servers C4140-M and R940xa with NVIDIA GPUs. We ran multiple tests to cover several configurations such as single node and multi node as well as storing data on local disk vs using NFS (network file system). We also investigated how NVIDIA’s implementation of RAPIDS memory manager helps in the overall speedup. [2].

The main objective is to demonstrate how to speed up machine learning workflows with the RAPIDS accelerated software stack, increasing performance in terms of productivity and accuracy at a lower infrastructure cost.
1 RAPIDS Overview

RAPIDS is a GPU accelerated data science pipeline, and it consists of open-source software libraries based on python to accelerate the complete workflow from data ingestion and manipulation to machine learning training. It does this by:

1. Adopting the columnar data structure called GPU data frame as the common data format across all GPU-accelerated libraries.
2. Accelerating data science building blocks (such as data manipulation, routines, and machine learning algorithms) by processing data and retaining the results in the GPU memory.

Figure 1 shows the main software libraries as part of RAPIDS:

- cuDF: Is the GPU DataFrame library with Pandas-like API style for data cleaning and transformation. It is a single repository containing both the low-level implementation and C/C++ API (LibGDF) and high-level wrappers and APIs (PyGDF). It allows to convert Pandas DataFrame to GPU DataFrame (Pandas ↔ PyGDF)
- cuML: Suite of libraries with the implementation of machine learning algorithms compatible with RAPIDS ecosystem; including Clustering, Principal Components Analysis, Linear Regression, Logistic Regression, XGBoost GBDT, XGBoost Random Forest, K-Nearest Neighbors (KNN), GLM (including Logistic), Support Vector Machines, among others.
- cuGraph: Library for Graph Analytics

Figure 1. RAPIDS open Source Software. Source: Nvidia
Data Processing Evolution:
In a benchmark consisting of aggregating data, the CPU becomes the bottleneck because there is too much data movement between the CPU and the GPU. So, RAPIDS is focused on the full data science workflow and keeping data on the GPU (using same memory format as Apache Arrow). As you lower data movement between CPU & GPU, it leads to faster data processing as shown in Figure 2.

![DATA PROCESSING EVOLUTION](image)

Figure 2. Data Processing Evolution. Source: Nvidia

Pillars of Rapids Performance:
- CUDA Architecture: Massively parallel processing
- NVLink/NVSwitch: High speed connecting between GPUs for distributed algorithms
- Memory Architecture: Large virtual GPU memory, high speed memory

1.1 XGBoost

XGBoost is one of the most popular machine learning packages for training gradient boosted decision trees. Native cuDF support allows to pass data directly to XGBoost while remaining in GPU memory. Its popularity relies on its strong history of success on a wide range of problems and for being the winner of several competitions, increasing the stakeholder confidence in its predictions. However, it has some known limitations such as the tradeoff of scale out and accuracy and issues with considerable number of hyperparameters can take long time to find the best solution. Figure 3 shows the average ranking of the ML algorithms and the XGBoost is one of the leading algorithms.
1.2 Dask for Parallel Computing

Dask is a distributed computation scheduler built to scale Python workloads from laptops to supercomputer clusters. It takes advantage of hardware advancements using a communications framework called OpenUCX to scale up and out with cuDF (Native integration with Dask + cuDF):

- For intranode data movement, utilizing NVLink and PCIe peer-to-peer communications
- For internode data movement, utilizing GPU RDMA over InfiniBand and RoCE
1.3 Why RAPIDS, Dask and XGBoost?

There are several reasons to bring together these tools:

- Freedom to execute end-to-end data science & analytics pipelines entirely on GPU
- User-friendly Python interfaces
- Relies on CUDA primitives
- Faster results make tuning parameters more interactive, leading to more accuracy in predictions and therefore more business value
- Dask provides advanced parallelism for data science pipelines at scale. It works with the existing Python ecosystem to scale it to multi-core machines and distributed clusters, sharing their syntaxes
- cuML also features multi-GPU and multi-node-multi-GPU operation, using Dask
- XGBoost takes advantage of fast parallel processing with GPUs in both single and multi-node configurations to reduce training times

1.4 New York City (NYC) – Taxi Dataset

Description:
The yellow taxi trip records include fields capturing pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts.

Source:
The data used in the datasets were collected and provided to the NYC Taxi and Limousine Commission (TLC) by technology providers authorized under the Taxicab & Livery Passenger Enhancement Programs (TPEP/LPEP). The trip data was not created by the TLC, and TLC makes no representations as to the accuracy of these data.

Size:
The dataset used in this project contains historical records accumulated and saved on individual monthly files from 2014 to 2016 (Total: ~64GB), with the below sizes per year:

![NYC Taxi Dataset Size per Year (GB)](image)

Figure 5. NYC Taxi Dataset Size (GB)
1.5 E2E NYC-Taxi Notebook

This is an End to End (E2E) notebook example extracted from Nvidia rapids ai/notebooks-contrib GitHub repo, the workflow consists of three core phases: Extract-Transform-Load (ETL), Machine Learning Training, and Inference operations performed on the NYC-Taxi dataset. The notebook focuses on showing how to use cuDF with Dask & XGBoost to scale GPU DataFrame ETL-style operations & model training out to multiple GPUs on multiple nodes. see below Figure 6. In this notebook we will see how RAPIDS, Dask, and XGBoost are implemented to work together.

![Figure 6. NYC-Taxi Notebook Workflow](image)

1.6 RAPIDS Memory Manager (RMM)

According to Nvidia definition “RAPIDS Memory Manager (RMM) is a central place for all device memory allocations in cuDF (C++ and Python) and other RAPIDS libraries. In addition, it is a replacement allocator for CUDA Device Memory (and CUDA Managed Memory) and a pool allocator to make CUDA device memory allocation / deallocation faster and asynchronous.”
2 System Configuration

Test System Hardware:
Servers
- C4140-M
  - 4xV100-SXM2-16GB
- R940xa
  - 4xV100-PCle-32GB
  - 4xV100-PCle-16GB
- Network connection over InfiniBand
- R740xd server hosting the NFS with the dataset for remote data

Test System Software:
- Ubuntu 18.04
  - Docker CE v19.03+ for Linux distribution
- RAPIDS Version: 0.10.0 - Docker Install
  - Nvidia Driver: 418.67+
  - CUDA: 10.1

Note: The systems were tested using RAPIDS via docker NVIDIA GPU Cloud (NGC); the original notebook requires Google DataProc to access the hosted NYC-Taxi Dataset; to bypass this dependency we downloaded the dataset locally and update the notebook to read it from a specific path.

Test Configuration:
The tests were conducted using the below variations:

- Using dataset from year 2014 until year 2016, exploring the maximum dataset capacity to be handled by each server.
- RMM enable and disable
- Single Node [3-4] and Multi Node [5-8]
- Local data and remote data on NFS [9]
- To ensure reproducibility, each server was tested 3 times and calculated the average as the result.
3 Results on Single Node

The following session shows the results on single node mode for each server tested.

2014 Year Dataset SATA Data vs NVMe Data:
We started with 2014-year dataset (26.5GB) using PowerEdge servers C4140-M with NVIDIA V100-SXM2-16GB and R940xa with NVIDIA V100PCle-16GB and V100PCle-32GB. The results shown below were conducted with the feature RMM disabled. See Figure 7.

![Performance on Dell EMC PowerEdge Servers in Single Node](image)

Figure 7. Performance on Dell EMC PowerEdge Servers in Single Node

RMM enable vs RMM disable:
On the server with 32GB device memory PowerEdge R940xa (4xV100PCle-32GB) with 2014-year dataset, the RMM feature was tested on enable and disable mode; RMM enable yielded the shortest total E2E time (~101 seconds), 19% faster than its configuration with RMM disable (~124 seconds). See Figure 8.

![Performance on Server R940xa with RMM enable vs RMM disable](image)

Figure 8. Performance on Server R940xa with RMM enable vs RMM disable
Remote data on NFS versus Local data on SATA:
Another aspect to explore was the effect of using remote data on NFS versus local data on SATA device. To do so, we tested on the server C4140-M 4xV100-SXM2-16GB with local data 2014-year on SATA drive, which was just 3% faster than remote data on NFS. See Figure 9

![Figure 9](image)

**Figure 9. Performance on Server C4140-M Comparing Remote Data on NFS vs Local Data on SATA**

Increasing the Dataset Size:
As we mentioned on the test configuration, the dataset size was gradually increased on a year basis and in this section we added 2015-year as well as 2016-year dataset to test each server's compute capability.

Server PowerEdge R940xa (4xV100PCle-32GB) with RMM enable handled 48.4GB data size
Server PowerEdge R940xa (4xV100PCle-32GB) with RMM enable handled up to 58.8 GB of data size

![Figure 10](image)

**Figure 10. Largest Dataset Handled by Server R940xa**
4 Results on Multi Node on C4140-M

To run RAPIDS in multi-node mode we used Dask CUDA to extend Dask distributed with GPU support. There are different methods to set up the multi-node mode depending on the target cluster, for more options see Dask documentation as reference [5-8]. In this case we will set up the cluster with 2 nodes:

- The primary compute node C4140-M server hosting the dask-scheduler
- Number of GPU's (workers) in primary compute node: 4
- Jupyter notebook on the primary node
- A secondary compute node C4140-M server with additional 4 GPU's (workers)
- Total GPU's in the cluster: 8
- R740xd server hosting the NFS with the dataset

Scale Out RAPIDS on C4140-M versus C4140-M Single Node:
The server C4140-M 4xV100-SXM2-16GB in multi node was tested first with 2014-year dataset only, the results were compared with its performance in single node to determine the workflow acceleration (with RMM disable in both cases); in multi node mode the system speeded up around 55% faster than single node. The main acceleration was reflected at the ETL phase, 99.5 seconds versus 50.4 seconds. See Figure 11 below.

Scale Out RAPIDS on C4140-M versus R940xa Single Node:
Since the server C4140-M in multi node has total device capacity of 128GB, we compared it versus the server R940xa in Single Node which has the same total device capacity of 128GB; in this case the servers were configured with RMM enable. We found that although both systems have the same total device capacity, the C4140-M multi node performed 58% faster than R940xa single node. The faster speed-up times is based on the number of GPUs allocated i.e. C4140-M with 8x GPUs in multi-node vs R940xa with 4x GPUs. See Figure 12.
Scale Out RAPIDS on C4140-M | Faster Performance on Largest Dataset:
In this section, we tested the system C4140-M 4xV100-SXM2-16GB in multi node, increasing the data size gradually by year and month. The system has the capacity to handle up to 51.7GB data size, the highest dataset processing capacity (2014-2015-2016 Jan-Feb dataset with 51.7GB) in the shortest total E2E time (126 seconds). Figure 13 summarize the maximum data processing capacity exposed for each server tested on this project. The winner was C4140-M in multi node.
5 Results with System Profile in “Performance” Mode

In order to boost the performance, the System Profile was changed under the Bios Settings from “Performance per Watt (DAPC)”, the default configuration used to run the previous tests, to “Performance” mode; as a result, the performance was boosted between 7% - 9% in terms of Total E2E seconds. For instance, the E2E on PowerEdge C4140-M-4xV100-SXM2-16GB in Multi Node went from 59 seconds to 55 seconds (7% faster), see below Figure 14:

The servers C4140-M-4xV100-SXM2-16GB and PowerEdge R940xa (4xV100PCIe-32GB) were tested on performance mode also and single node, presenting 9% of performance boosting each, see Figure 15 and Figure 16:

![Figure 14. C4140-M-4xV100-SXM2-16GB Multi Node | Performance vs Performance per Watt mode](image-url)

![Figure 15. C4140-M-4xV100-SXM2-16GB | Single Node | Performance vs Performance per Watt mode](image-url)
Figure 16. R940xa (4xV100PCle-32GB) | Single Node | Performance vs Performance per Watt
6 Conclusion and Future Work

We have shown how Dell EMC PowerEdge servers with NVIDIA GPUs can be used to accelerate your data science pipeline with RAPIDS. We have compared performance using both NVIDIA NVLINK & PCIE GPUs using scale-up and scale-out server’s solutions using different storage configurations.

Main highlights:

- Using NYC-taxi 2014 dataset, server C4140-M 4V100-SXM2-16GB in Multi Node mode (8x16=128GB) with RMM enable yielded the shortest total E2E time (~59 seconds), 58% faster than server R940 4xV100PCle-32GB Single Node (~101 seconds)
- C4140-M 4xV100-SXM2-16GB with local data on SATA drive was 3% faster than remote data on NFS
- System Profile Settings in “Performance” mode yield ~7%-9% in boost performance

The experiments run in this paper show the basic method to deploy RAPIDS with DASK on multiple nodes. As alternative to automate the deployment for multi-node in production environments, the tests can be conducted using a cluster resource manager such as SLURM, PBS, Kubernetes, Yarn among others.
A Dell EMC PowerEdge Server Specifications

The below table shows the technical specifications of the servers used in this paper

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>Dell EMC PowerEdge</td>
<td>Dell EMC PowerEdge</td>
<td>R940xa</td>
<td>R940xa</td>
</tr>
<tr>
<td></td>
<td>C4140 Conf. M</td>
<td>C4140 Conf. M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU Model</td>
<td>Intel(R) Xeon(R) Gold</td>
<td>Intel(R) Xeon(R) Gold</td>
<td>Intel(R) Xeon(R) Gold</td>
<td>Intel(R) Xeon(R) Gold</td>
</tr>
<tr>
<td></td>
<td>6148 CPU @ 2.40GHz</td>
<td>6148 CPU @ 2.40GHz</td>
<td>Platinum 8180M CPU @</td>
<td>Platinum 8180M CPU @</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.50GHz</td>
<td>2.50GHz</td>
</tr>
<tr>
<td>System Memory</td>
<td>512 GB</td>
<td>256 GB</td>
<td>3022 GB</td>
<td>3022 GB</td>
</tr>
<tr>
<td>DIMM</td>
<td>64GB</td>
<td>16GB</td>
<td>64GB</td>
<td>64GB</td>
</tr>
<tr>
<td></td>
<td>M386A8K40BMB-CPB</td>
<td>M393A2K43BB1-CTD</td>
<td>HMAA8GL7AMR4N-VK</td>
<td>M386A8K40BM2-CTD</td>
</tr>
<tr>
<td>Storage</td>
<td>1) SATA Disk (M.2)</td>
<td>1) SATA Disk (M.2)</td>
<td>PCIe Data Center SSD</td>
<td>PCIe Data Center SSD</td>
</tr>
<tr>
<td></td>
<td>2) NVMe-PCIe adapter</td>
<td>2) NVMe-PCIe adapter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPUs</td>
<td>4 x V100-SXM2-16GB</td>
<td>4 x V100-SXM2-16GB</td>
<td>4 x V100-PCIe-16GB</td>
<td>4 x V100-PCIe-32GB</td>
</tr>
<tr>
<td>Total GPU memory</td>
<td>64GB</td>
<td>64GB</td>
<td>64GB</td>
<td>128GB</td>
</tr>
<tr>
<td>Nvidia Driver</td>
<td>418.87</td>
<td>418.67</td>
<td>418.56</td>
<td>418.67</td>
</tr>
<tr>
<td>CUDA</td>
<td>10.1</td>
<td>10.1</td>
<td>10.1</td>
<td>10.1</td>
</tr>
<tr>
<td>OS</td>
<td>Ubuntu 18.04.1</td>
<td>Ubuntu 16.04.6 LTS</td>
<td>Ubuntu 18.04.2 LTS</td>
<td>Ubuntu 18.04.2 LTS</td>
</tr>
<tr>
<td>Kernel</td>
<td>GNU/Linux 4.15.0-66-</td>
<td>GNU/Linux 4.4.0-143-</td>
<td>GNU/Linux 4.15.0-51-</td>
<td>GNU/Linux 4.15.0-66-</td>
</tr>
<tr>
<td></td>
<td>generic x86_64</td>
<td>generic x86_64</td>
<td>generic x86_64</td>
<td>generic x86_64</td>
</tr>
</tbody>
</table>
**Terminology**

**B Terminology**

**RAPIDS**: Suite of software libraries, built on CUDA-X AI, that gives the freedom to execute end-to-end data science and analytics pipelines entirely on GPUs

**End to End workflow**: Data science pipeline that includes the three phases of ETL (Extract, Transform, Load), data conversion, and training

**Dask**: Open source freely available that provides advanced parallelism for analytics. It is developed in coordination with other community projects like Numpy, Pandas, and Scikit-Learn

**XGBoost**: Open-source software library which provides a gradient boosting framework for C++, Java, Python, R, and Julia. It works on Linux, Windows, and macOS

**Docker mounted volume**: An existing directory on the host that is “mounted” to be available inside the container, useful for sharing files between the host and the container

**Cluster**: Group of computers communicating through fast interconnection

**Node**: Group of processors communicating through shared memory

**Socket**: Group of cores communicating through shared cache

**Core**: Group of functional units communicating through registers

**Pipeline**: Sequence of instructions sharing functional units

**Threads**: The smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system

**Dask-scheduler**: Coordinates and execute the task graphs on parallel hardware

**Dask-worker**: Computes tasks as directed by the schedules, stores and serves computed results to other workers or clients

**Dask-cuda**: Allows deployment and management of Dask workers on CUDA-enabled systems

**Diagnostic dashboard**: Interactive dashboard containing several plots and tables with live information about task runtimes, communication, statistical profiling, load balancing, memory use, and so on.

**Bokeh**: Interactive visualization library that targets modern web browsers for presentation
Example; GPU Activity with C4140-M in Multi Node Mode

C Example; GPU Activity with C4140-M in Multi Node Mode

GPU Activity on C4140-M in Multi Node Mode with RMM enable
Example; GPU Activity with C4140-M in Multi Node Mode

GPU Activity on C4140-M in Multi Node Mode with RMM disable
**D  Dask Diagnostic Dashboard**

Dask diagnostic dashboard helps to understand the performance of the code running on the cluster among each worker, please watch the video “Dask Dashboard walkthrough” for detailed explanation of each dashboard page [12].

Task and thread activities among the workers over the time. The workers are identified with horizontal bars, in this example there are four workers.
Profile page allows to inspect the code performance at the finest granularity level, each horizontal bar corresponds to a function.

The System page provides plots with information about the resource utilization when the scheduler runs processes.
Workers page provides information about all the workers running on the cluster.

![Workers page screenshot](image1)

Dashboard – Workers page

Info page provides more information about each worker running on the cluster. It provides log files for each worker and the log file for the scheduler, allowing to inspect a particular task, its dependencies, memory resources used, and its progress through the scheduler.

![Info page screenshot](image2)

Dashboard – Info page
E  NVDashboard – Nvidia GPU Dashboard

As an alternative monitoring tool, NVDashboard is an open-source package for the real-time visualization of NVIDIA GPU metrics on interactive Jupyter environments. The dashboards use pynvml to access GPU information attached to the machine and display the plots in Jupyter Lab environment.

**Instructions to install NVDashboard on docker**

Run the docker image:

```
$ docker run --gpus all --rm -it --net=host -p 8888:8888 -p 8787:8787 -p 8786:8786
rapidsai/rapidsai:0.10-cuda10.1-runtime-ubuntu18.04
```

Once inside the docker, add GPU Dashboards with the below commands:

```
$ pip install jupyterlab-nvdashboard
$ jupyter labextension install jupyterlab-nvdashboard
```

Start Jupyter notebook, it will run at the designated output IP/Port

```
$ bash utils/start-jupyter.sh
```

Once the NVDashboard is installed, the “System Dashboard” will be visible at the left side of the Jupyter-Lab environment, see below some plot samples.
GPU Dashboards

GPU Utilization

GPU Resources
PCIe Throughput

Machine Resources
Environment set up

In this section we explain the steps on how to install RAPIDS through the docker from NVIDIA GPU Cloud (NGC), download the NYC-taxi dataset, and pull the notebooks repo [13]

1. Review the below prerequisites before running the tests:
   a. NVIDIA Pascal™ GPU architecture or better
   b. CUDA 9.2 or 10.0+ compatible NVIDIA driver
   c. Ubuntu 16.04/18.04 or CentOS 7
   d. Docker CE v19.03+ for Linux distribution

2. Download and the NYC-taxi dataset to the folder of your choice at the local host, for example:
   $ mkdir rapids cd rapids
   $ mkdir data cd data

3. Pull the notebooks-contrib repo inside the container using wget or to the local host and use a docker volume mount to /rapids/contrib/:
   $ cd rapids
   $ git clone https://github.com/rapidsai/notebooks-contrib

   **Note:** Use the dataset and notebooks-contrib paths as the mounted volumes, as per the instructions running docker images. This will map folders from the host operating system to the container OS in the /rapids/ directories

4. Pull the selected docker image from NGC. To explore the full tag list for all available images visit:
   $ docker pull nvcr.io/nvidia/rapidsai:rapidsai:0.10-cudal0.1-runtime-ubuntu18.04

5. Start the Container:
   $ docker run --gpus all
   --rm -it \
   --net=host \ 
   -p 8888:8888 \ 
   -p 8787:8787 \ 
   -p 8786:8786 \ 
   -v /rapids/notebooks-contrib/:/rapids/notebooks/contrib/ \ 
   -v /rapids/data/:/rapids/data/ \
   nvcr.io/nvidia/rapidsai:rapidsai:0.10-cudal0.1-runtime-ubuntu18.04
G Notebook NYC-Taxi Set Up

See below the steps to start the notebook server and the notebook example

1. Once within the container, start the Notebook Server on the host machine (this will run JupyterLab on port 8888 on the host machine):

   ```
   (rapids) root@container:/rapids/notebooks# bash utils/start-jupyter.sh
   ```

   **Note:** To run JupyterLab on a different port, edit and modify the start-jupyter.sh file as below adding the flag `--port=<another_port>`, and re-start the Notebook Server:

   ```
   jupyter lab --allow-root --ip=0.0.0.0 --no-browser --NotebookApp.token="" --port=<another_port>
   ```

2. To access Jupyter, open a browser with the url address:

   ```
   http://<IP_local_host>:<port>/
   ```

   The nyc-taxi notebook can be found in the following directory:

   ```
   rapids/notebooks/contrib/intermediate_notebooks/E2E/taxi/NYCTaxi_E2E.ipynb
   ```

3. Modify the NYCTaxi_E2E.ipynb notebook and provide the data path in the volume mounted previously:

   ```
   base_path = '/home/dell/rapids/data/nyc-taxi/'
   ```

4. To run the data set on a specific year, proceed to comment the cells aimed to increase the data size and limit the DataFrame to that year, example:

   ```
   Limit the dataset to a specific year:
   taxi_df = dask.dataframe.multi.concat([df_2015])
   ```

   ```
   Include multiple years:
   taxi_df = dask.dataframe.multi.concat([df_2014, df_2015, df_2016])
   ```
### RAPIDS Multi Node Set Up

1. **Run as Docker container on each node**
   
   On each node, go inside the RAPIDS docker image and start the multi-node configuration as described in the next steps. Below is the command example to go within the docker:

   ```bash
docker run --runtime=nvidia 
   --rm -it --net=host 
   -p 8888:8888 
   -p 8787:8787 
   -v /home/rapids/notebooks-contrib:/rapids/notebooks/contrib/ 
   -v /home/rapids/data:/home/dell/rapids/data/ 
   nvcr.io/nvidia/rapidsai/rapidsai:0.10-cuda10.1-runtime-ubuntu18.04
```

2. **Launch the dask-scheduler on the primary compute node**

   ```bash
   $ dask-scheduler --port=8888 --bokeh-port 8786
   output:
   distributed.scheduler - INFO - Receive client connection: Client-9ad22140-83bd-11e9-823c-246e96b3e316
   distributed.core - INFO - Starting established connection
   ```

3. **Launch dask-cuda-worker on the primary compute node**

   This step will start workers at the same Primary machine as the scheduler was started

   ```bash
   $ dask-cuda-worker tcp://<ip_primary_node>:8888
   output:..... messages with successful connection
   ```

4. **Launch dask-cuda-worker on the secondary compute node**

   This step will start additional workers on the secondary compute node

   ```bash
   $ dask-cuda-worker tcp://<ip_primary_node>:8888
   output:..... messages with successful connection
   ```

5. **Start Jupyter and run the notebook (client python API) on the primary compute node**

   In this case, the NYC-Taxi notebook is the Client Python API which will be attached to the scheduler running on the primary compute node, so it can be run using all compute node GPUs in distributed mode. To do so, we need to modify the notebook, starting the client and providing the primary node IP and port designated to be listened as below:

   ```python
   client = Client('tcp://<ip_primary_node>:8888')  # connect to cluster
   output:
   ```

---

**Client**

- Scheduler: tcp://<ip_primary_node>:8888
- Dashboard: http://<ip_primary_node>:8786/status

**Cluster**

- Workers: 8 # total workers in distributed mode
- Cores: 8
- Memory: 67.47 GB
I  Bios Settings to Boost Performance

Bios Settings to Boost Performance

![Integrated Dell Remote Access Controller 9 Express](image)

- **System Profile Settings**
  - **Current Value**
    - **System Profile**: Performance
    - **CPU Power Management**: Maximum Performance
    - **Memory Frequency**: Maximum Performance
    - **Turbo Boost**: Enabled
    - **C1E**: Disabled
    - **C States**: Disabled
    - **Write Data CRC**: Disabled
    - **Memory Patrol Scrub**: Standard
    - **Memory Refresh Rate**: 1x
    - **Uncore Frequency**: Maximum
    - **Energy Efficient Policy**: Performance
    - **Number of Turbo Boost Enabled Cores for Processor 1**: All
    - **Number of Turbo Boost Enabled Cores for Processor 2**: All
    - **Number of Turbo Boost Enabled Cores for Processor 3**: All
    - **Number of Turbo Boost Enabled Cores for Processor 4**: All
    - **Monitor/Mwait**: Enabled
    - **Workload Profile**: Not Available
    - **CPU Interconnect Bus Link Power Management**: Disabled
    - **PCI ASPM L1 Link Power Management**: Disabled

[Apply] [Discard]
### Bios Settings to Boost Performance

![Integrated Dell Remote Access Controller 9 Express](image)

#### Processor Settings

<table>
<thead>
<tr>
<th>Feature</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical Processor</td>
<td>Enabled</td>
</tr>
<tr>
<td>CPU Interconnect Speed</td>
<td>Maximum data rate</td>
</tr>
<tr>
<td>Virtualization Technology</td>
<td>Enabled</td>
</tr>
<tr>
<td>Adjacent Cache Line Prefetch</td>
<td>Enabled</td>
</tr>
<tr>
<td>Hardware Prefetcher</td>
<td>Enabled</td>
</tr>
<tr>
<td>Software Prefetcher</td>
<td>Enabled</td>
</tr>
<tr>
<td>DCU Streamer Prefetcher</td>
<td>Enabled</td>
</tr>
<tr>
<td>DCU IP Prefetcher</td>
<td>Enabled</td>
</tr>
<tr>
<td>Sub NUMA Cluster</td>
<td>Disabled</td>
</tr>
<tr>
<td>UPI Prefetch</td>
<td>Enabled</td>
</tr>
<tr>
<td>Logical Processor Idling</td>
<td>Disabled</td>
</tr>
<tr>
<td>Configurable TDP</td>
<td>Nominal</td>
</tr>
<tr>
<td>x2APIC Mode</td>
<td>Enabled</td>
</tr>
<tr>
<td>Dell Controlled Turbo</td>
<td>Disabled</td>
</tr>
<tr>
<td>Dell AVX Scaling Technology</td>
<td>0</td>
</tr>
<tr>
<td>Number of Cores per Processor</td>
<td>All</td>
</tr>
<tr>
<td>Processor Core Speed</td>
<td>2.10 GHz</td>
</tr>
<tr>
<td>Processor Bus Speed</td>
<td>10.40 GT/s</td>
</tr>
<tr>
<td>Family-Model-Stepping</td>
<td>6-66-6</td>
</tr>
<tr>
<td>Brand</td>
<td>Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz</td>
</tr>
</tbody>
</table>

RAPIDS Scaling on Dell EMC PowerEdge Servers
J Common Errors

During the tests we experimented GPU device memory issues, for more details on the memory performance and issues we encountered please see the section A “Controlling memory usage”. These errors have been documented and explained by NVIDIA [10] as below:

“Running out of GPU Device Memory:

- ETL processes may create many copies of data in device memory, resulting in memory utilization spikes
- Need to budget 25% GPU device memory to account for XGBoost overhead
- Cannot exceed 24GB on 32GB GPU, or cannot exceed 12GB on 16GB GPU
- Memory utilization which exceeds available device resources will cause a Dask worker to crash
- This error can be propagated forward in the Dask task graph, and manifest in very short ETL times (sub-millisecond timescale)
- An error may be raised by another routine referring to None Type in data or similar

Running out of system memory:

- “The final step of the ETL process migrates all computed results back to system memory before training, and if you do not have sufficient system memory, your program will crash. The step before training migrates a portion of the data back into device memory for XGBoost to train against”
K  Technical Resources

https://rapids.ai/

K.1  Related Resources


• [6] XGBoost with Rapids – Nvidia webinar


• [10] Common Errors https://docs.rapids.ai/containers/rapids-demo#common-errors


© 2019 Dell Inc. or its subsidiaries. All Rights Reserved. Dell, EMC and other trademarks are trademarks of Dell Inc. or its subsidiaries. Other trademarks may be trademarks of their respective owners.