RAPIDS

Scaling Up And Out in Python

Optimizing 1TB+ Datasets on Distributed GPU Systems with Dask and RAPIDS

Nick Becker and Ben Zaitl

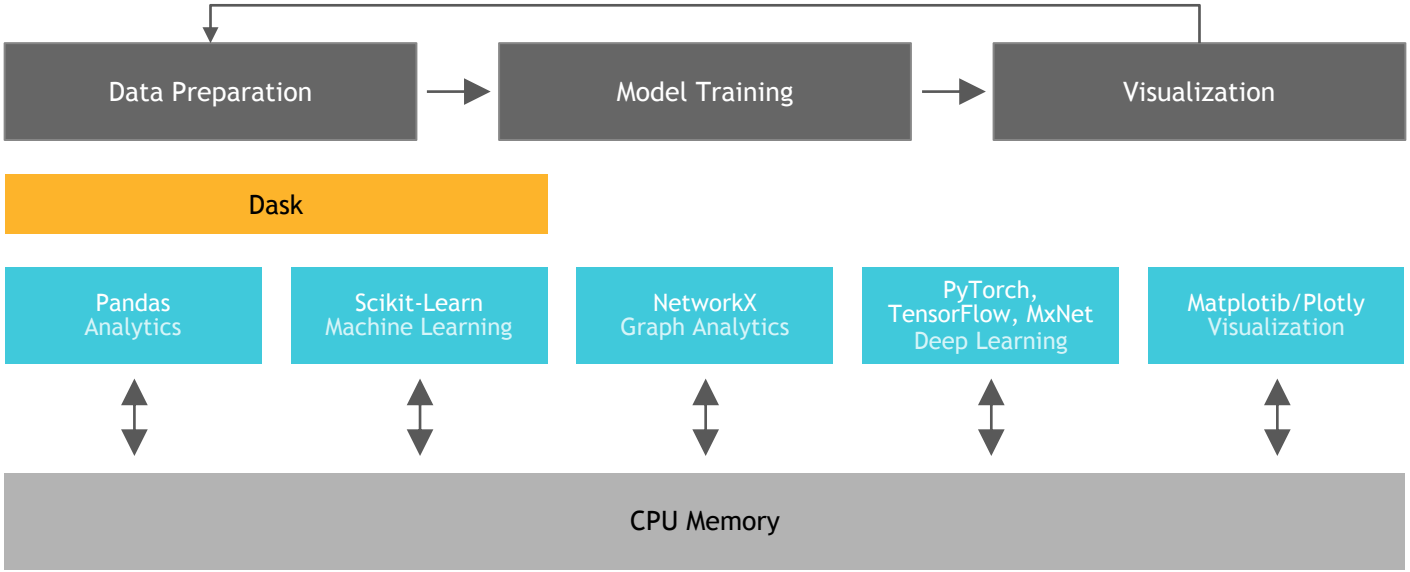
Agenda

- RAPIDS / Dask / UCX Overview
- RAPIDS TPCx-BB
- New Tooling for New Challenges
- Development Spotlight
- Key Learnings
- Next Steps

RAPIDS / Dask / UCX Overview

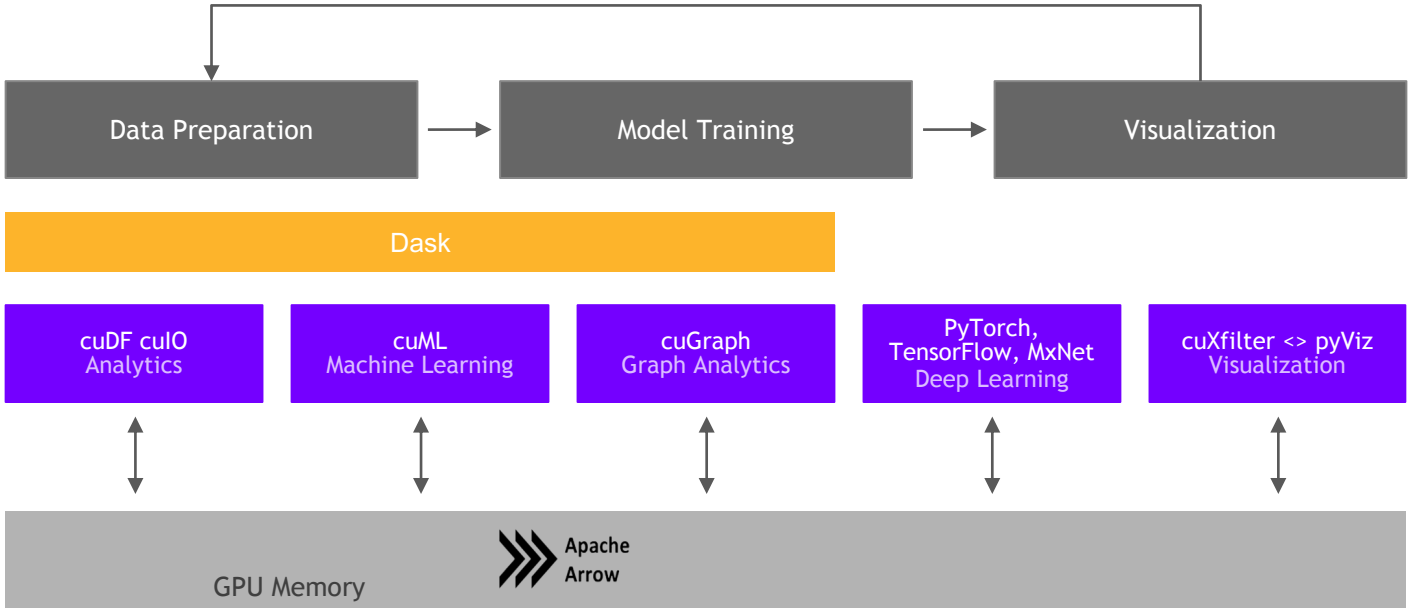
Open Source Data Science Ecosystem

Familiar Python APIs



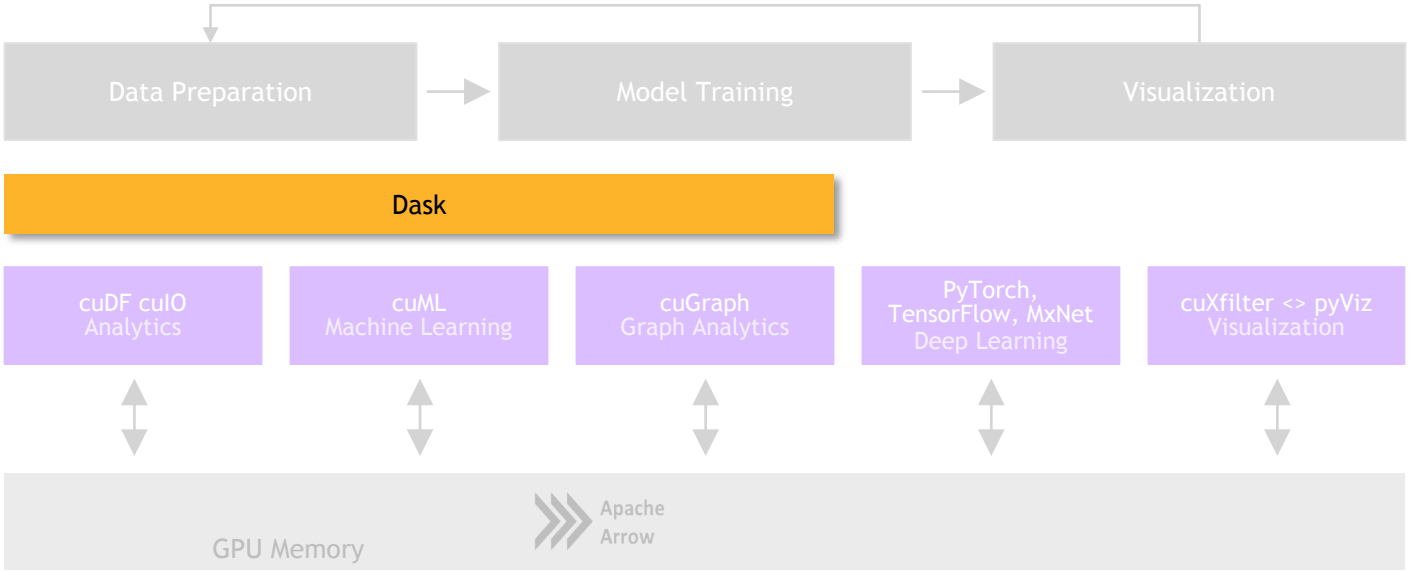
RAPIDS

End-to-End Accelerated GPU Data Science



RAPIDS

Scaling RAPIDS with Dask



What is Dask?

- Distributed compute scheduler built to scale Python
- Scales workloads from laptops to supercomputer clusters
- Extremely modular: disjoint scheduling, compute, data transfer and out-of-core handling
- Multiple workers per node allow easier one-worker-per-GPU model

Why Dask?

DEPLOYABLE

- **HPC:** SLURM, PBS, LSF, SGE
- **Cloud:** Kubernetes
- **Hadoop/Spark:** Yarn

PYDATA NATIVE

- **Easy Migration:** Built on top of NumPy, Pandas, Scikit-Learn, etc
- **Easy Training:** With the same APIs
- **Trusted:** With the same developer community

EASY SCALABILITY

- Easy to install and use on a laptop
- Scales out to thousand node clusters

POPULAR

- Most Common parallelism framework today in the PyData and SciPy community

Scale-Out Problem

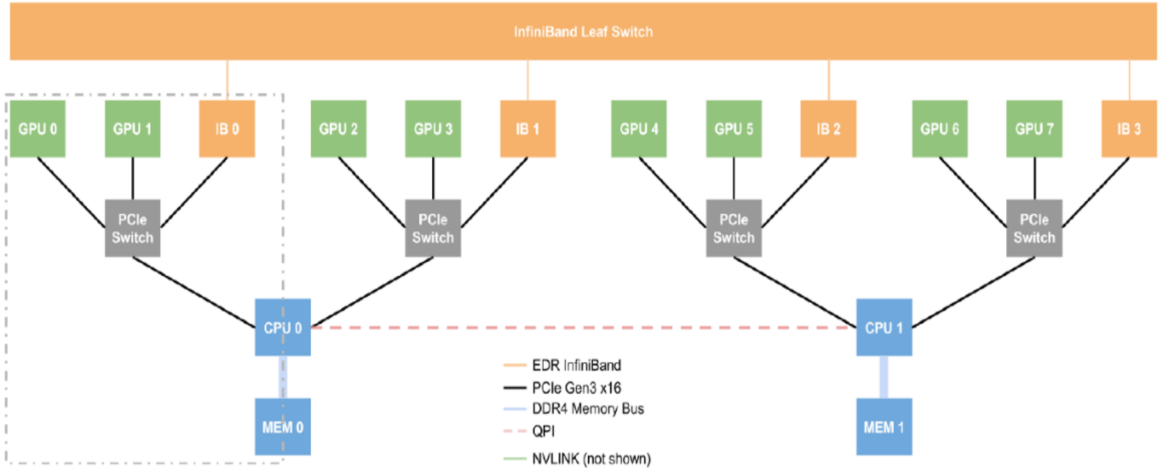
Networking

Scale-out bottlenecks occur at the networking layer

We need to efficiently pass data and minimize HtoD/DtoH transfers

For DGX-1, what is the best path to pass data between GPU0 and GPU5

Available comms: TCP, NVLink, InfiniBand



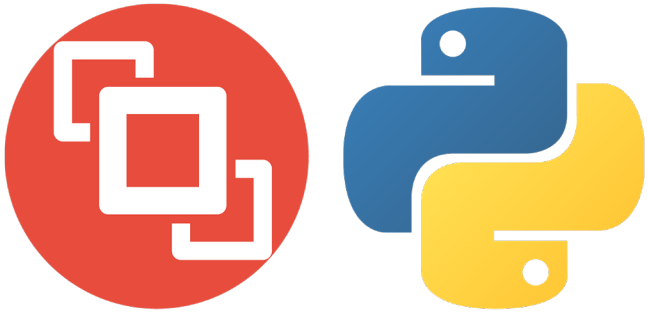
Scale-Out Solution

Networking

OpenUCX is a low-latency high-bandwidth library with support for traditional and accelerated networking hardware

Recently created UCX-Py library allows Dask to expose accelerated networking to Distributed PyData

Adding Python layers has minimal impact on performance

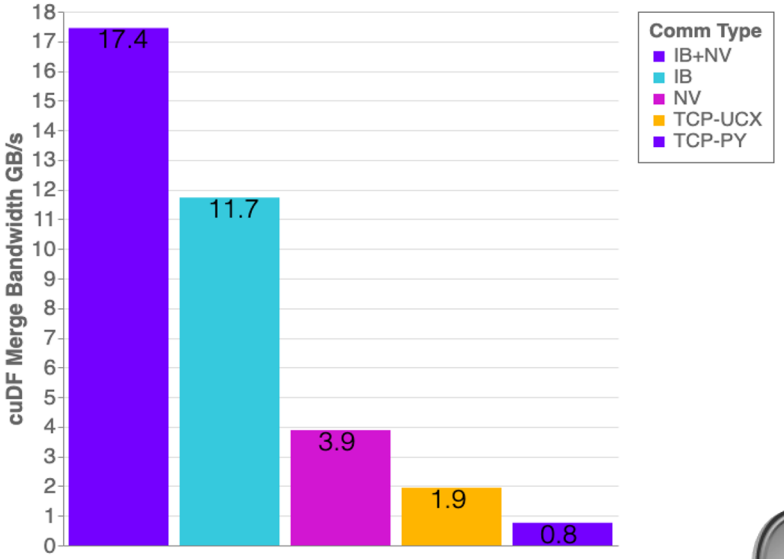


1GB Messages	Theoretical	UCX-Py
NVLink	50 GB/s	46.5 GB/s
InfiniBand	12.5GB/s	11.2 GB/s

Benchmarks: Distributed cuDF Random Merge

cuDF v0.14, UCX-PY 0.14

- Running on NVIDIA DGX-1:
  - GPU: NVIDIA Tesla V100 32GB
  - CPU: Intel(R) Xeon(R) CPU 8168 @ 2.70GHz
- Benchmark Setup:
  - DataFrames: Left/Right 1x int64 column key column, 1x int64 value columns
  - Merge: Inner
  - 30% of matching data balanced across each partition



Scale Out with RAPIDS + Dask with OpenUCX

Scale Up / Accelerate

RAPIDS AND OTHERS

Accelerated on single GPU

NumPy -> CuPy/PyTorch/..
Pandas -> cuDF
Scikit-Learn -> cuML
Numba -> Numba

RAPIDS

RAPIDS + DASK WITH OPENUCX

Multi-GPU
On single Node (DGX)
Or across a cluster

RAPIDS

DASK

PYDATA

NumPy, Pandas, Scikit-Learn, Numba and many more

Single CPU core
In-memory data

Numba NumPy pandas PyData learn

DASK

Multi-core and distributed PyData

NumPy -> Dask Array
Pandas -> Dask DataFrame
Scikit-Learn -> Dask-ML
… -> Dask Futures

DASK

Scale Out / Parallelize

RAPIDS TPCx-BB

What is TPCx-BB®?

Comparing Big Data Platforms since the Cambrian Explosion of Big Data

- TPCx-BB is a Data Analytics and Data Science benchmark. It is a common set of stylized business use-cases to compare various technologies in an apples-to-apples way

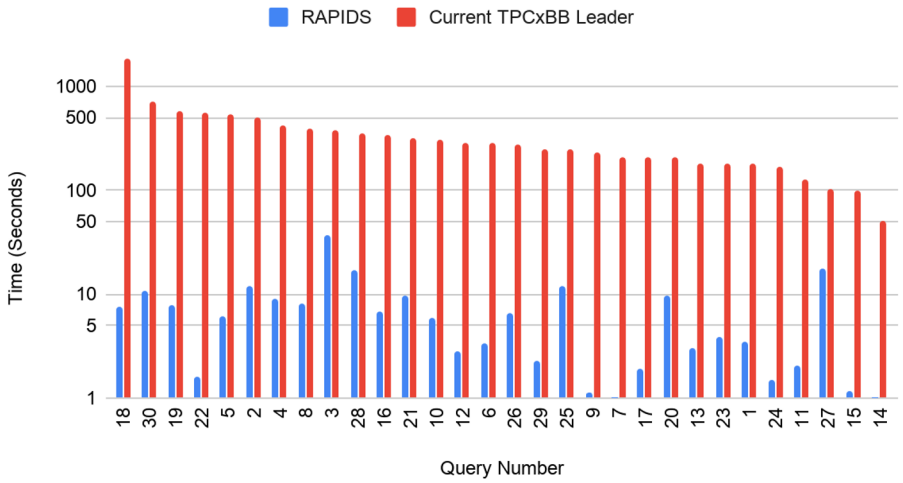
- 30 frequently performed analytical queries in the context of retailers with physical and online store presence.

- Query Examples
  - Build a model for a visitor to an online store based on activity and demographics
  - Find the top 100 products frequently sold together in selected stores

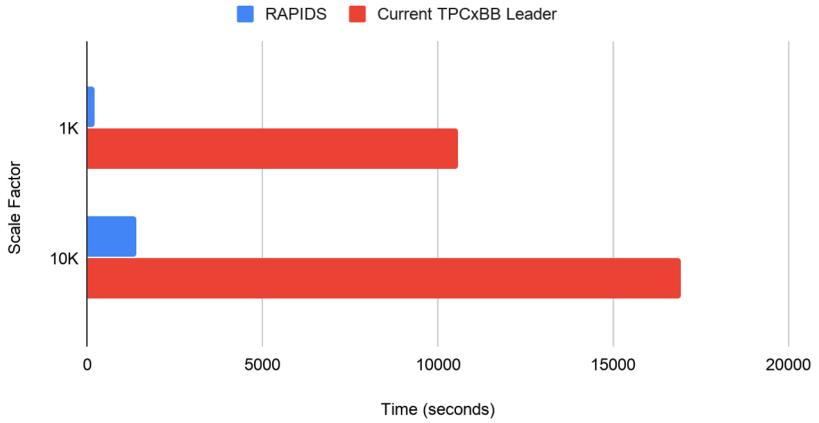
RAPIDS Results

Up to 350x Faster

SF1K Speedup with RAPIDS



TPCxBB Total Time RAPIDS vs Current Leaders



TPCx-BB Setup

Nuts and Bolts

1TB and 10TB Scale Factors

Raw CSVs

Convert to Parquet

Experiments with 1GB, 2GB, and 3GB Partitions

Experiments with Filesystems: NFS/Lustre/Weka

SF1K: DGX2

16 GPUs fully connected with NVLink (NVSwitch) 512 GB Mem

2 Xeon Platinum

1.5 TBs Host Memory

10 InfiniBand Networking Devices

SF10K: 17 DGX1 Cluster (Per Node Details)

8 GPUs with NVLink 256 GB Mem

2 Xeon E5-2600

0.5 TB Host Memory

4 InfiniBand Networking Devices

Query Spotlight - NLP at Scale

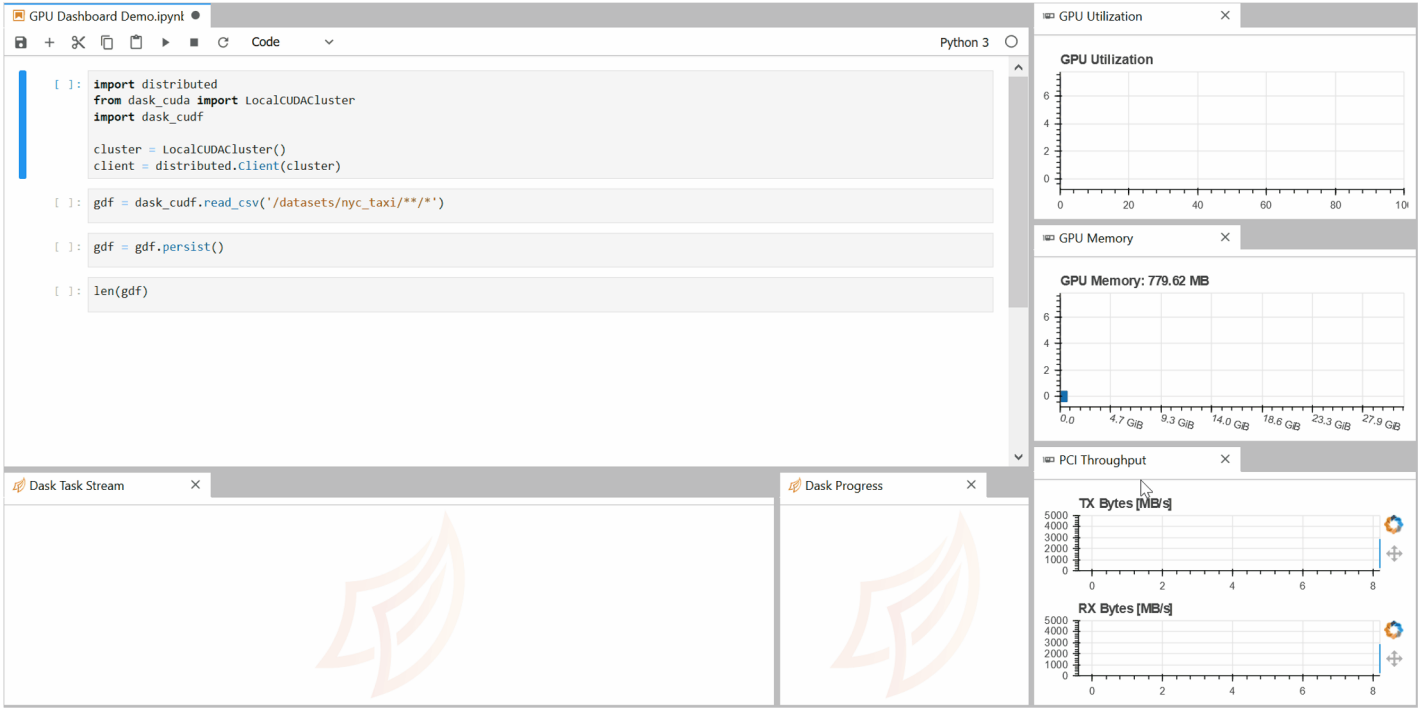
Query 18 - are bad reviews correlated with bad sales ?

- Read, filter to a set of four months, and merge the data
  - Read parquet files (split by row groups), filter to a date range, and joining several tables containing store, store sales, date, and customer review data
- For each store, calculate the linear regression of total net sales over time and select those stores with a negative slope
- Find reviews that include any of those store names
- For reviews that contain a store's name, return sentences containing a negative word and the negative word itself
  - Tokenize reviews into sentences and word to search sentences for words contained in a text file of negative words
  - Return the store name, date of the review, sentence, and word for sentences where negative words appeared

New Tooling for New Challenges

Development Environment

Adapt Existing Tools



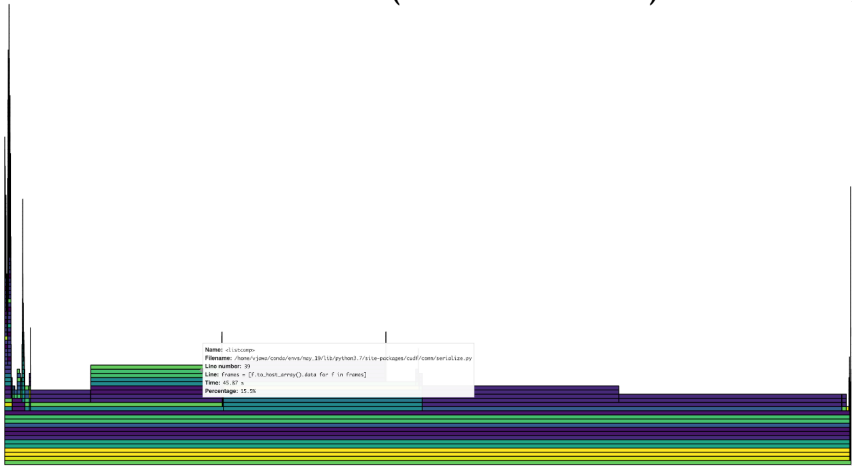
- Jupyter Lab
- Dask Extension
- NVDashboard Extension

Dask Debugging

Dask Performance Report

Task Stream

Worker Profile (Administration)



Find where are workflows spending time

Fun Game: Reduce and/or eliminate bars

Multi-GPU Debugging

NSight Systems Profiles

Find where are workflows spending time

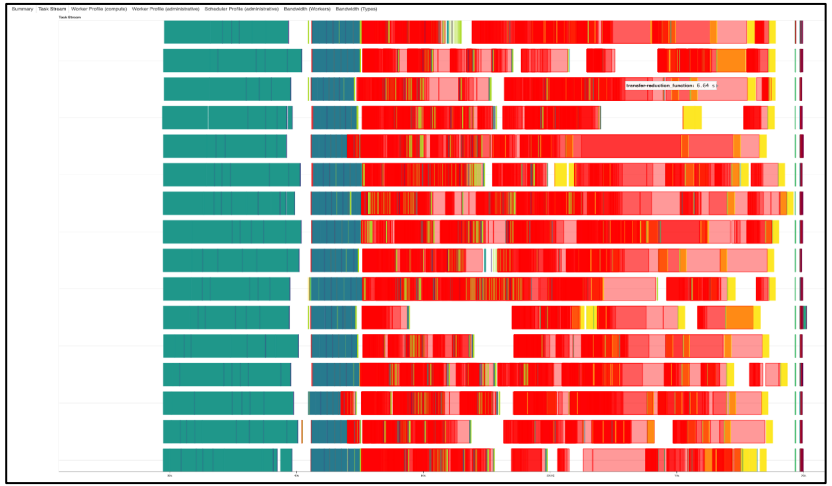
Not Just Multi-GPU but Multi-Process

Fun Game: Reduce and/or eliminate bars

Network Performance

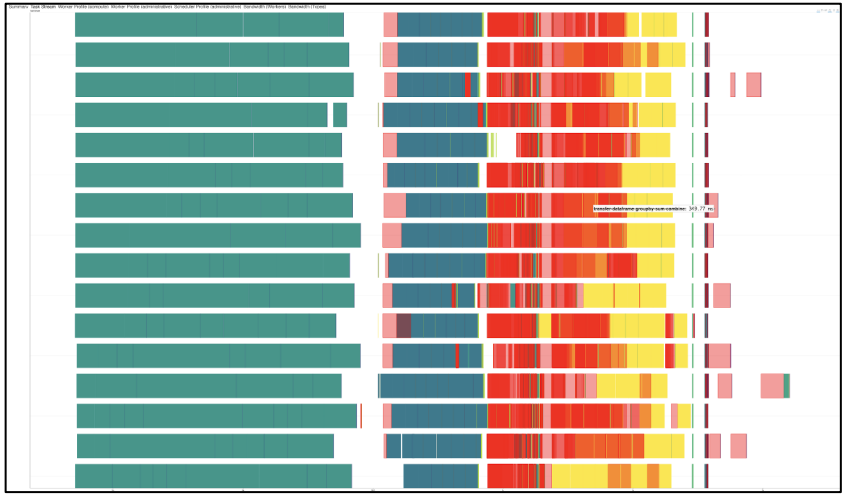
UCX (NVLink) vs TCP

TCP



64.8 Sec

UCX



43.2 Sec

Red Is Transfer, Other colors are Tasks.

Development Spotlight

RAPIDS Development

- Improved Ecosystem Interoperability and Memory Management
  - GPU Memory Management Plugins
- DataFrames
  - New Groupby Functionality
  - Multi-column quantiles and improved sorting
  - Left-semi joins
  - Improved API consistency with Pandas
  - Optimized concatenate
- I/O
  - GPU-accelerated parquet readers/writers
- Machine Learning
  - Hashing Vectorizer
  - Naive Bayes Classifier
  - Improved K-Means Clustering
  - New GPU-accelerated metrics

Dask Development

- DataFrames
  - Generalized more functions to dataframe-like objects and improved support for cuDF DataFrames
  - Column based hash-repartitioning
  - Multi-column sorting (GPU-only, for now)
  - Left-semi joins
  - Improved resiliency of cumulative aggregations
- Arrays
  - Significantly enhanced support for sparse GPU arrays
- I/O
  - Support for the GPU-accelerated parquet readers/writers
- Administration
  - Defined and improved serialization protocols for GPU objects

UCX Development

- Serialization
  - Ensure proper serialization from cuDF through Dask
- InfiniBand/NVLink
  - Use a Memory Pool
  - Map Memory Pool Once Between Workers
- Dask
  - Easy Replacement for TCP
  - Consolidate Buffers Before Transfer (ongoing work)

Running RAPIDS TPCx-BB At 1TB and 10 TBs

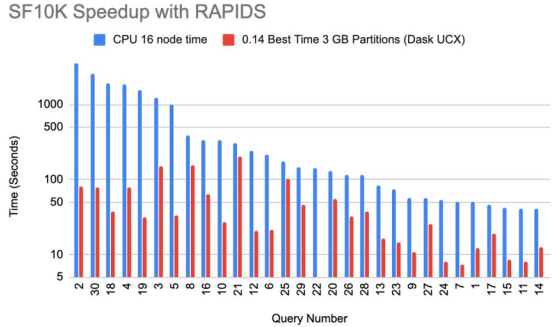
Up to 350x faster queries

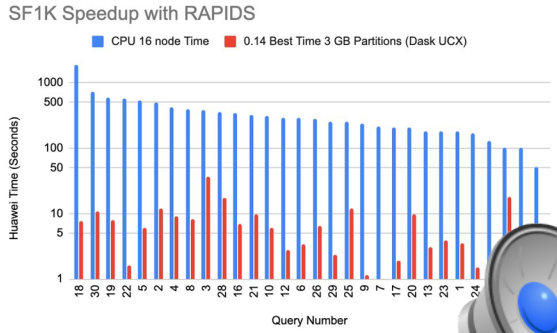
Like other TPC benchmarks, TPCx-BB can be run at multiple “Scale Factors”:

- SF1 - 1GB
- SF1K - 1 TB
- SF10K - 10 TB

We’ve been benchmarking RAPIDS implementations of the TPCx-BB queries at the SF1K (Single DGX-2) & SF10K (17x DGX-1) scales

Our results indicate that GPUs provide dramatic cost and time-savings for small scale *and* large-scale data analytics problems





Key Learnings

Key Workflow Learnings

Research to Production

- Partition Size Matters
  - Larger partitions improve GPU utilization and reduce scheduler overhead
- Memory Pressure is Real at Scale
  - Downcast types where appropriate
  - Avoid materialized indexes (after boolean indexing)
  - Adding explicit synchronizations (wait) can help reduce memory pressure for very large workflows
  - Fill nulls to avoid storing the null mask
- Avoid Shuffle-Joins
  - Convert shuffle-joins into broadcast-joins where possible
- I/O
  - Fusing reading and filtering tasks releases memory faster and prevents spilling related overhead
  - Convert to a single partition after I/O when you have small dataframes
  - Split by row groups for better parallelism

Next Steps

Faster Faster Faster

Current Bottlenecks

Disk IO	More Networking Acceleration	Core Dask Components
Currently, reading data from disk requires a mem-mapped read on the host and copy from host memory into GPU memory This adds latency and increases host memory requirements The upcoming CUDA cuFile API enables a direct to GPU memory data path between both local and remote NVME storage systems This greatly increases throughput for IO heavy jobs Bypass host memory and read directly onto GPU -- **GPU Direct Storage**	Current TPCx-BB results use UCX with NVLink but not yet InfiniBand with GPU RDMA Detection of high speed networking hardware like NVLink and InfiniBand Even in the absence of accelerated networking hardware, Dask can now leverage lower level networking primitives instead of slower Python native TCP sockets	RAPIDS team continues to contribute back to the Dask open-source project, and Dask will continue to be key to using RAPIDS at scale. As Dask improves, we expect our numbers to improve as well

Additional Resources

RAPIDS

https://rapids.ai/start

RAPIDS TPCxBB

https://github.com/rapidsai/tpcx-bb

Dask / Dask-CUDA

dask.org
https://dask-cuda.readthedocs.io/en/latest/

UCX / UCXPy

https://www.openucx.org
https://ucx-py.readthedocs.io/en/latest/index

Ben Zaitlen
bzaitlen@nvidia.com

Nick Becker
nicholasb@nvidia.com

RAPIDS