Daskify an MPI application for distribution using Dask

Learnings from implementation

Sangeeth Keeriyadath, Pradipta Ghosh

IBM

SciPy2020

Scientific Computing with Python
Virtual Conference • July 6-12

Background : Advancement of H/W and ML frameworks

- **Hardware design (e.g. IBM Power System AC922)**

  Summit and Sierra remain in the top three spots. IBM-built supercomputers employing Power9 CPUs and NVIDIA Tesla V100 GPUs.

- **ML library utilize advances in hardware and algorithms (e.g. Snap ML)**
  - Scale out "Distributed training" implementation for massive datasets (Supports MPI and Spark)
  - Specialized solvers designed for "GPU acceleration"
  - Optimized algorithms for "Sparse data structures"

Motivation : HPC infrastructure inhibitors for Data Scientists

Python is the de facto ecosystem for data scientists and MPI is the proven performer in HPC world for decades.

MPI adoption inhibitors in data science world :

- Over head of switching between different languages for existing libraries.
- MPI has lack of integration with popular IDEs, web interfaces and tools.

To handle Big Data in Pythonland - DASK was born!

The journey to excel leveraging distributed computing continues ...

.	MPI	hadoop	Spark	DASK	.	RAY	MODIN

Goal :

Use Dask distributed processing for data exploration and feature engineering

feed into

State-of-the-art distributed machine learning library **SnapML** for training

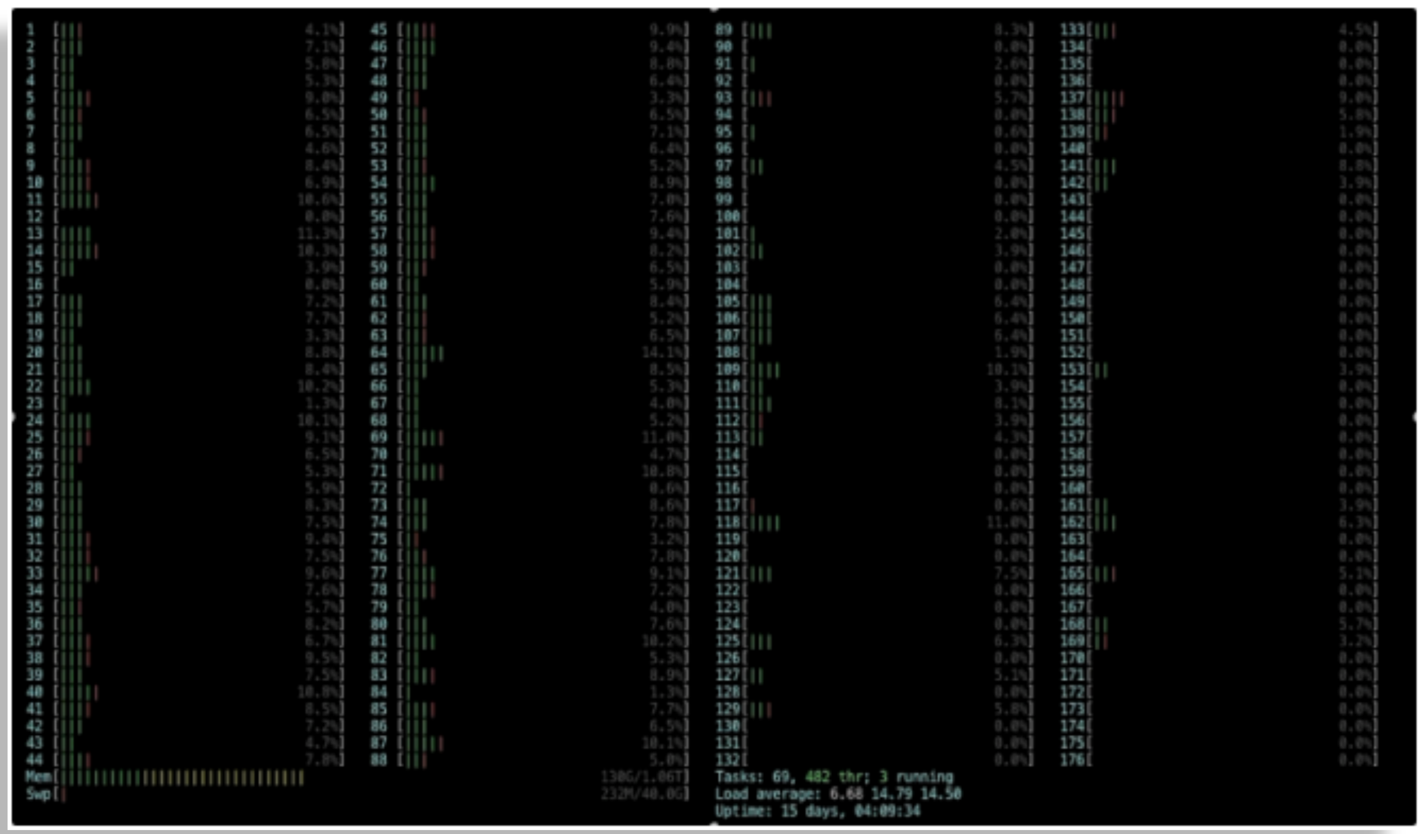
"JupyterLab and its offshoots are the most common, with 83% of data scientists using it on a regular basis."

Design considerations

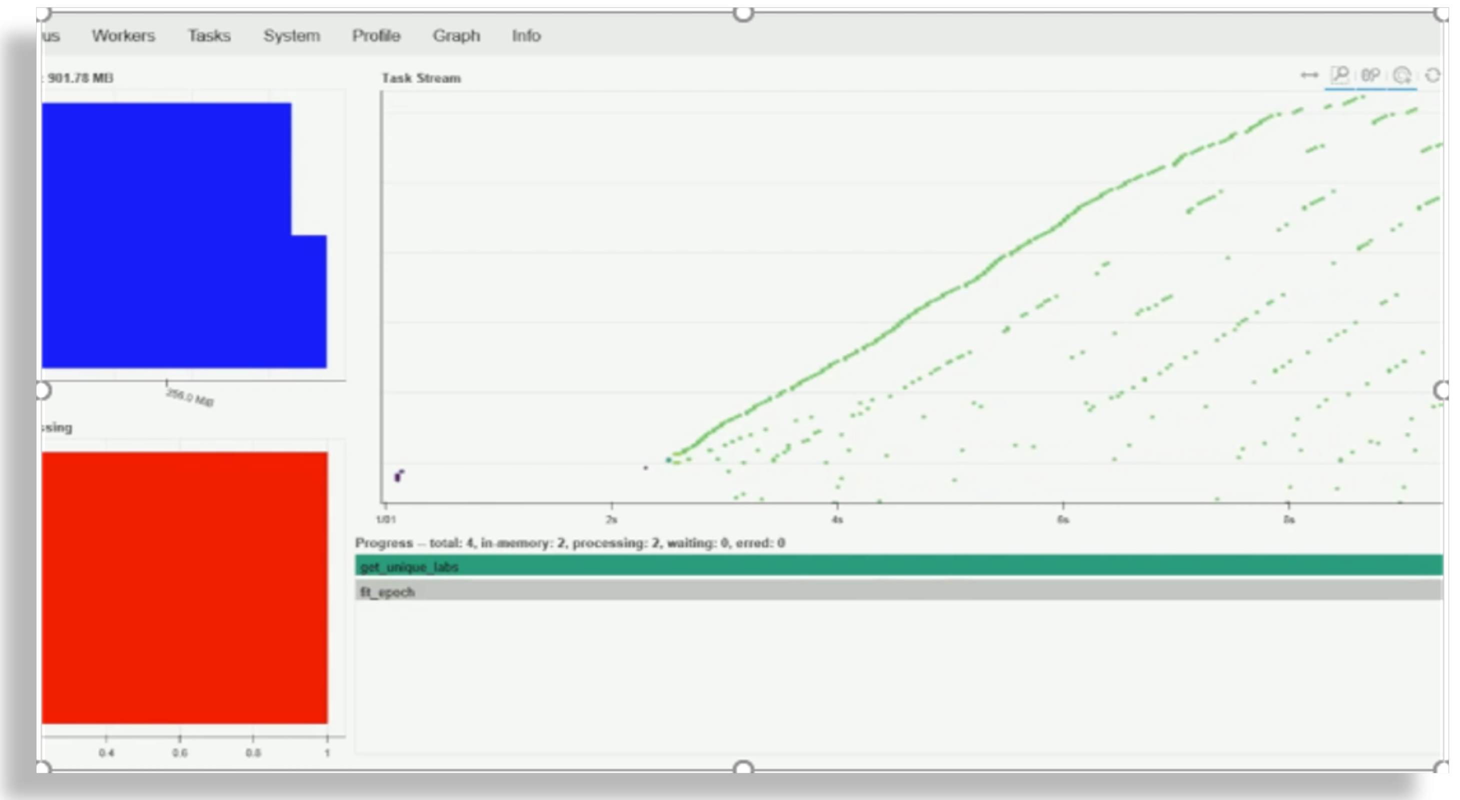
- ❑ Redesign code flow to make best use of dask future objects for parallel processing
- ❑ Use one chunk of daskarray per node of the cluster (use `X.rechunk()`)
- ❑ Make use of dask primitives to perform in-place global reduction operations
  - `MPI_SUM` to compute total number of positive labels was replaced with `da.sum(y > 0 ).compute()`
  - `MPI_Send / MPI_Recv` and reduction on master was replaced with Collecting result of each chunk from the future objects, and using python `functools.reduce()` to aggregate those results
- ❑ For the portions of processing in the C++ library, extract numpy array from individual chunks of dask array (use `X.compute()`)

11

CPU Usage : `htop` view of all the cores engaged with processing dask workload



Dashboard : Daskified SnapML - What's Say



Highlights :

- Replaced MPI operations with Dask Distribution framework APIs for pure pythonic experience in Jupyter Notebook
- Achieving the same result with ~10% overhead compared to pure MPI based distribution (*not that bad, ugh!*)

Path ahead :

- Add optimization in pythonic substitute of MPI reduction operations.
- Use this changes for next generations CPU's like s390x

Special Thanks!

Work powered by SnapML from IBM Research - Zurich

Reference links :

- https://www.top500.org/lists/top500/2020/06/
- https://www.zurich.ibm.com/snapml/
- https://dask.org/