How to do Machine Learning on Big Cluster

Thomas J. Ashby, Tom Vander Aa
IMEC
Belgium
Email: {ashby, tom.vanderaa}@imec.be

Stanislav Böhm, Vojtěch Cima, Jan Martinovič
IT4Innovations
Czech Republic
Email: {first.last}@vsb.cz

Vladimir Chupakhin
Jannsen R&D
Belgium
Email: vchupakh@its.jnj.com

I. Introduction

Design and development of new drugs is a complex and a very expensive procedure. Academia and pharma industry is extensively trying to reuse the data obtained during the previous experiments to cut the costs, time and animal usage. Various approaches can be applied to model the interaction of the chemical compound with the biological organism and most importantly the targets of the chemical compounds. The most common method is to find a relationship between the structure of the compound and its biological activities via regression or classification. Systematic exploration of those quantitative structure-activity relationships often called chemogenomics. Its major requirement is diverse input data e.g. ChEMBL[1], a public resource, contains 1.5 million of compounds and 14 million of compound-activity endpoints on 11000 activity types. An industry scale data is significantly larger and highly imbalanced toward inactive compounds. Unfortunately, in both cases the known compound-target-activity triplets are limited, giving us very sparse known interaction space, where modeling can help to find new insights.

Computational chemogenomics in pharma industry can be treated as a large scale machine learning for highly imbalanced dataset with heterogeneous data representation. Optimal model will require testing of several machine learning algorithms and several representations of the chemical compounds. Stringent nested cross-validation setup is required to avoid overfitting of the models; hyperparameters search will add additional computational runs. For example, 5 by 5 nested cross-validation setup with 10 runs on average to estimate optimal hyperparameters and 5 type of compound representations will result in 5 x 5 x 10 x 5 = 1025 computational runs, taking into account all targets can give us 11 275 000 runs per one machine learning method. A task that is hardly feasible for desktop computational environment.

In European H2020 project ExCAPE[1], we are investigating how best to use extreme scale machines to tackle this challenge, dealing with aspects such as algorithm design, software development and testing on realistic industrial data sets using supercomputers. As such this project bridges HPC and Big Data challenges, concerning the calculation of complex empirically driven models on large data sets.

[1]http://www.excape-h2020.eu/

II. HyperLoom

One of basic challenges when applying machine learning in HPC environment on a large scale is managing relatively small interdependent tasks. To deal with this problem, we are developing HyperLoom[2] – an open source framework for defining and executing scientific pipelines. HyperLoom provides the following features:

- In-memory data processing reducing filesystem load.
- Direct worker-to-worker data transfer reducing server overhead.
- Support for execution of third party applications.
- Support for short and long tasks (from tens of milliseconds to hours).
- Data-location aware scheduling algorithm reducing internode network traffic.
- C++ core with a Python client enabling high performance through a simple API.
- High scalability and native HPC support.

A. Related works

PBS[3] and TORQUE[4] represent common HPC scheduler to provide fair and secure resource sharing among users of an HPC system. These schedulers aim for relatively long running tasks and do not allow express interdependency.

Although other workflow frameworks for data processing like Luigi[5], Airflow[6], Pinball[7], Pegasus[2] exist. These tools serve for building, managing, and visualization of pipelines of batch jobs. From perspective of our problem domain, tasks in these tools interchange data through a shared file system that contains a major bottleneck for executing a large number of small tasks.

As far as we know, Dask/Distributed[8] is the most similar project to HyperLoom. The main difference is the choice of programming languages for each of them. While HyperLoom's performance-critical components are implemented in C++ and Python is only used on client side, Dask/Distributed is completely implemented in Python which introduces performance bottleneck for large-scale Dask/Distributed deployments.

[2]https://code.it4i.cz/ADAS/loom
[3]http://www.pbsworks.com/
[4]http://www.adaptivecomputing.com/products/open-source/torque/
[5]https://github.com/spotify/luigi
[6]https://github.com/airbnb/airflow
[7]https://github.com/pinterest/pinball
[8]https://github.com/dask/distributed

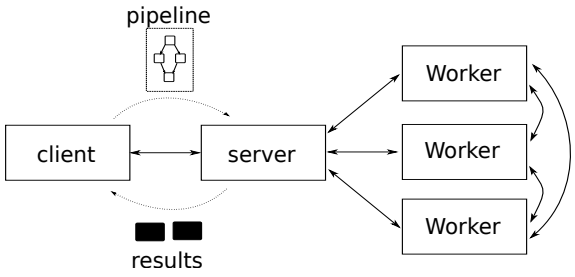

Fig. 1. Loom architecture.

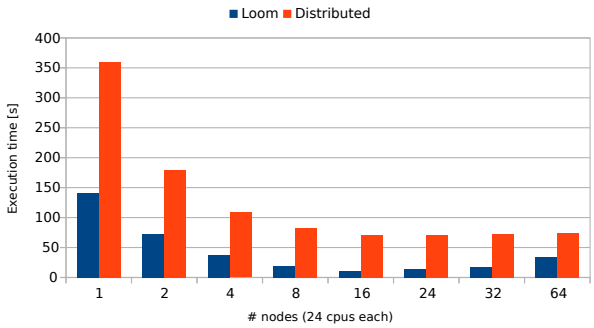

Fig. 2. Comparison of the plan execution time [s] in HyperLoom and Dask/Distributed (*50kh*).

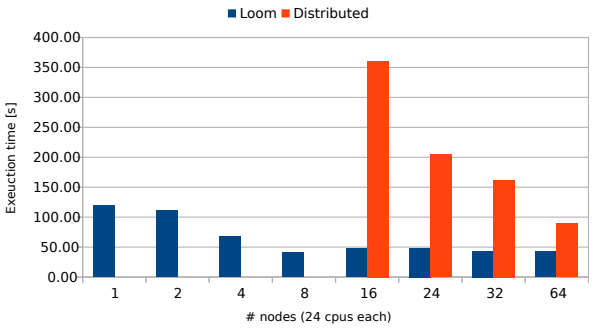

Fig. 3. Comparison of the plan execution time [s] in HyperLoom and Dask/Distributed (*gridcat*).

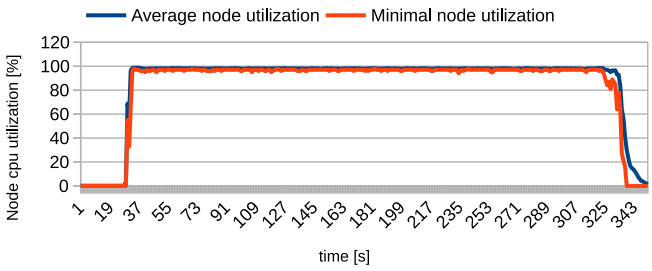

Fig. 4. Average and minimal node CPU on a benchmark derived from *mlchemo*, ~460k tasks; 32 nodes (24 cpus each)).

B. Architecture

Figure 1 shows the main components of HyperLoom. HyperLoom consists of *server*, a set of *worker* and *client* components providing the following functionality:

- **client** (Python) – allows users to programmatically chain computational tasks into a pipeline and submit the pipeline to server. It also provides a functionality to gather results of the submitted tasks after computation finishes.
- **server** (C++) – receives and decomposes a plan and reactively schedules tasks to run on available computational resources provided by workers.
- **workers** (C++) – execute and run tasks as scheduled by server and inform the server about the state of the task states.

HyperLoom components can be deployed on various types of large-scale distributed systems. In context of HPC centers, HyperLoom components can be deployed on resources (compute nodes) allocated by lower level schedulers such as TORQUE or PBS. The whole pipeline is then processed within a single job.

III. Benchmarks

A. Scheduling Overhead

Here we contrast the scheduling overhead in HyperLoom and Dask/Distributed by comparing the plan execution times of both for the *50kh* test case which is composed mostly of independent short running tasks. Thus, *50kh* is expected to stress the scheduling process allowing us to analyze the scheduling overhead. Figure 2 shows execution time of *50kh*.

B. Scheduling Quality

In some cases, tasks generate significant amount of data. As a consequence, a suboptimal task placement results in delays due to data transfer between workers. In this regard, we have designed the *gridcat* test case to simulate this type of scenarios. Figure 3 shows execution times of the benchmark.

C. Cluster Utilization

The following benchmark shows a CPU utilization on a benchmark directly derived from *mlchemo* – chemogenomics machine learning pipeline used in the ExCAPE project.

Acknowledgement

This work was supported was supported from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement no. 671555 (ExCAPE project) and The Ministry of Education, Youth and Sports from the Large Infrastructures for Research, Experimental Development and Innovations project IT4Innovations National Supercomputing Center – LM2015070

REFERENCES

[1] A. P. Bento, A. Gaulton, A. Hersey, L. J. Bellis, J. Chambers, M. Davies, F. A. Krger, Y. Light, L. Mak, S. McGlinchey, M. Nowotka, G. Papadatos, R. Santos, and J. P. Overington, "The chembl bioactivity database: an update," *Nucleic Acids Research*, vol. 42, no. D1, pp. D1083–D1090, 2014. [Online]. Available: http://nar.oxfordjournals.org/content/42/D1/D1083.abstract

[2] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, "Pegasus: a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015, funding Acknowledgements: NSF ACI SDCI 0722019, NSF ACI SI2-SSI 1148515 and NSF OCI-1053575. [Online]. Available: http://pegasus.isi.edu/publications/2014/2014-fgcs-deelman.pdf