Co-funded by the Horizon 2020 Framework Programme of the European Union

Grant agreement No. 671555

ExCAPE
Exascale Compound Activity Prediction Engine

Future and Emerging Technologies (FET)

Call: H2020-FETHPC-2014
Topic: FETHPC-1-2014
Type of action: RIA

Deliverable D2.13

Report: Scalability Report 1

HyperLoom Scalability

Due date of deliverable: 31.12.2017
Actual submission date: 26.01.2018

Start date of Project: 1.9.2015 Duration: 36 months

Responsible Consortium Partner: Intel
Contributing Consortium Partners: IT4I
Name of author(s) and contributor(s): Vojtěch Cima (IT4I)
Internal Reviewer(s): Tom Vander Aa (IMEC), Yves Vandriessche (Intel), Vladimir Chupakhin (JPNV)
Revision: V3

Project co-funded by the European Commission within Horizon 2020 Framework Programme (2014-2020)		
Dissemination Level		
PU	Public	PU
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Document revision tracking

This page is used to follow the deliverable production from its first version until it has been reviewed by the assessment team. Please give details in the table below about successive releases.

Release number	Date	Reason of this release and/or validation	Dissemination of this release (task level, WP/ST level, Project Office Manager, Industrial Steering Committee, etc)
0	25.07.2017	First version	CO
1	20.12.2017	Internal review	CO
2	14.01.2018	Improved version	CO
3	26.01.2018	Final version	PU

Glossary

No glossary is given.

Link to Tasks

Task number	Work from task carried out	Deviations from task technical content, with motivation and summary of impact
2.5.1	Scalability benchmarking	None

Contents

Executive Summary

This report presents scalability of HyperLoom[1] on a multi-node systems using two synthetic and one real-world scenarios. The synthetic pipelines demonstrates the scalability for corner cases, the real world pipeline demonstrate the scalability for machine learning based pipelines used in ExCAPE. All the experiments were performed on a physical testbed with up to 1,536 distributed CPU cores.

Dissemination and Exploitation

The scalability results were disseminated within the following conference paper:

- Vojtěch Cima, Stanislav Böhm, Jan Martinovič, Jiří Dvorský, Katerina Janurová, Tom Vander Aa, Thomas J. Ashby, Vladimir I. Chupakhin: HyperLoom: A Platform for Defining and Executing Scientific Pipelines in Distributed Environments. PARMA-DITAM@HiPEAC 2018: 1-6

and SC17 research poster session:

- Thomas Ashby, Tom Vander Aa, Stanislav Böhm, Vojtěch Cima, Jan Martinovič, Vladimir Chupakhin. Poster How To Do Machine Learning on Big Clusters nominated for Best Poster Award. SC17, Denver, USA, 2017.

[1] www.hyperloom.eu

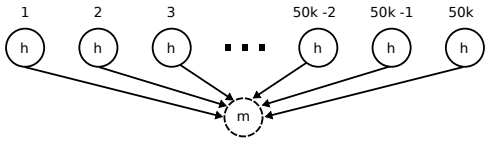


Figure 1: Shape of *50kh* pipeline.

1 Introduction

We evaluated HyperLoom performance and scalability through a series of experiments on a physical testbed. The scalability was measured based on the overall pipeline execution time – the period of time the pipelines took to complete using a different number of compute nodes. The elapsed time was measured from the time of pipeline submission to its successful completion. We also compare Hyperloom scalability to Dask/Distributed[2] that is a stardard tool for distributed execution of small to medium workloads.

2 Testbed

All the experiments have been performed on IT4Innovations' Salomon cluster using up to 64 dedicated identical physical computational nodes, each with two 12-core Intel Xeon E5-2680v3 processors (2.5GHz)[3] and 128 GB of physical RAM. The nodes are interconnected by 7D Enhanced hypercube Infiniband [4] network (56 Gbps). Nodes run Red Hat Enterprise Linux 6.5 [2].

3 Test Scenarios

We have designed three test cases. Two synthetic, devoted to evaluate performance of HyperLoom in comparison to Dask/Distributed, and one derived from a compound-activity modeling pipeline to demonstrate the scalability of HyperLoom for real-world application.

3.1 Scenario 1: 50kh

A synthetic test case designed to generate intensive scheduling load. The assembled pipeline contains 50k independent tasks that each executes the *hostname* program. Since this program completes instantly, it forces the scheduler to react promptly in order to keep workers utilized.

3.2 Scenario 2: gridcat

A synthetic test case designed to evaluate scheduling quality. The assembled pipeline contains 40 tasks that each generate 200MB of data, followed by a layer of 1,600 tasks that represent concatenations of every possible pair of the data generated in the first layer,

[2] https://distributed.readthedocs.io/en/latest/

[3] $http://ark.intel.com/products/81908/Intel-Xeon-Processor-E5-2680-v3-30M-Cache-2_50-GHz$

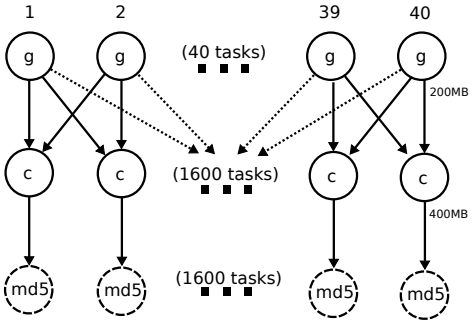


Figure 2: Shape of *gridcat* pipeline.

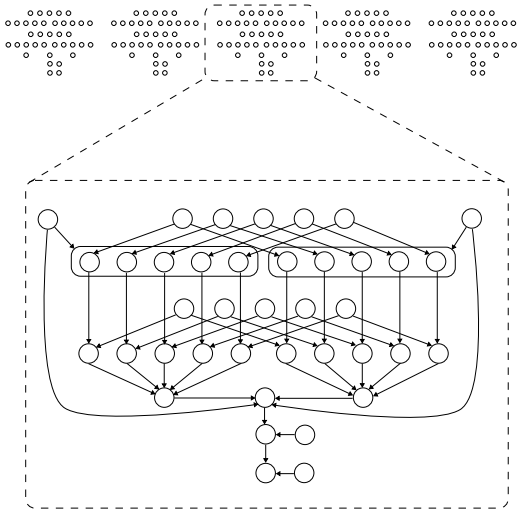


Figure 3: Shape of *mlchemo* pipeline.

followed by a layer of another 1,600 tasks that compute *md5* hashes of the concatenated data. If the scheduler does not utilize the location of the data, it will induce a significant inter-worker data transfer.

3.3 Scenario 3: mlchemo

A test case derived from a production pipeline used in ExCAPE. This pipeline performs a nested 5×5 cross-validation with a hyper-parameter search for machine-learning based models capturing compound-activity prediction. The shape of this pipeline is similar to the one depicted in Figure 3. The pipeline contains a mix of long running tasks such as modeling and validation done by LibSVM [1] – a widely used support vector machine implementation and a set of short running tasks providing auxiliary functionality.

4 Experiments

In this section we look at the scalability of the aforementioned scenarios.

Table 3: Pipeline execution time [s] in HyperLoom (HL) and Dask/Distributed (DD) (*50kh*, *gridcat*)

		50kh		*gridcat*	
# workers	# cores	HL	D/D	HL	D/D
1	24	141.48	359.00	119.78	N/A
8	192	19.66	81.91	40.47	N/A
16	384	11.24	71.03	47.72	360.72
32	768	17.41	73.10	43.42	162.00
64	1,536	34.28	73.80	41.98	89.45

4.1 Experiment 1: Scheduling Overhead

We contrast the scheduling overhead in HyperLoom and Dask/Distributed by comparing the pipeline execution time of the *50kh* test case that contains large number of independent short running tasks. Thus, *50kh* is expected to stress the reactive scheduling process which allows us to analyze the scheduling overhead.

Table 3 compares execution time of *50kh* using both, HyperLoom and Dask/Distributed, executing the pipeline on 1, 8, 16, 32, and 64 workers. In all of the cases, HyperLoom significantly outperforms Dask/Distributed completing the pipeline in less than half of the time. As the pipeline only contains independent short running tasks, we argue that the performance difference in this case is caused by the higher scheduling overhead.

4.2 Experiment 2: 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. We measure the execution time of *gridcat* in both, HyperLoom and Dask/Distributed using 1, 8, 16, 32, and 64 computational nodes. While HyperLoom successfully completes the pipeline execution in all of the test scenarios, the Dask/Distributed implementation fails when using less than 16 nodes due to an out-of-memory error. In all of the cases when both of the implementations finish, HyperLoom significantly outperforms Dask/Distributed. The exact figures for this experiment can be found in Table 3.

4.3 Experiment 3: Machine Learning Pipeline

We evaluate HyperLoom scalability using the *mlchemo* test scenario executing the pipeline on 1, 8, 16 and 64 workers (24 CPU cores each) and measure the total execution time for each. We demonstrate the performance for both, weak and strong scaling.

For the strong scaling experiments, we only increase the number of workers while keeping the pipeline size constant (˜460k tasks). For the weak scaling experiments, we linearly increase the pipeline size with the increasing number of workers by replicating a base *mlchemo* pipeline (˜12.5k tasks × # workers).

We compute strong scaling efficiency (SSE) as follows

$$SSE = \frac{t_1}{N t_n}, \tag{1}$$

Table 4: HyperLoom strong and weak scaling (*mlchemo*).

		Strong Scaling		Weak Scaling	
# workers	# cores	t [s]	SSE	t [s]	WSE
1	24	29,363	1.00	334	1.00
8	192	3,576	1.03	338	0.99
16	384	1,817	1.01	351	0.95
32	768	1,020	0.90	368	0.91
64	1,536	559	0.78	374	0.89

where t_1 is the execution time running on a single worker, N is the number of workers and t_n is the execution time running on N workers.

We compute weak scaling efficiency (WSE) as follows

$$WSE = \frac{t_1}{t_n}, \tag{2}$$

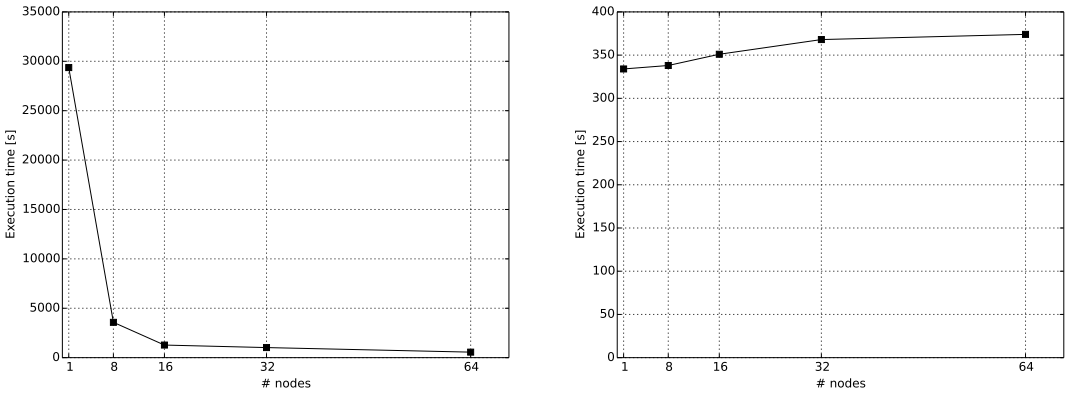
where t_1 is the execution time running on one worker, and t_n is the execution time running on N workers.

In Table 4, we observe almost linear decrease of the execution time with the increasing number of workers. Concretely, the execution time decreases from more than 8 hours (1 worker) to less than 10 minutes (64 workers). The SSE values are derived from the respective execution times using equation 1. Although, in the long run, the SSE decreases with the increasing number of workers, the observed decrease is very moderate; from SSE 1.0 (1 worker) to SSE $\approx$ 0.8 (64 workers). It is noteworthy that for the cases with 8 and 16 workers we even observe SSE to be slightly higher than 1.

Table 4 also shows the execution time for the weak scaling experiments where the number of tasks employed in the pipeline increases linearly with the number of workers. Ideally, we would expect that the execution time remains constant for all of the experiments. Nevertheless, increasing the pipeline and cluster size also increases the overall system overhead, which causes the execution time to increase. In particular, the execution time increases from 334 seconds (1 worker, $\approx$ 12.5k tasks) to 374 seconds (64 workers, $\approx$ 800k tasks). The WSE values are derived from the respective execution times using equation 2. WSE slightly decreases from 1 (1 worker) to $\approx$ 0.9 (64 workers).

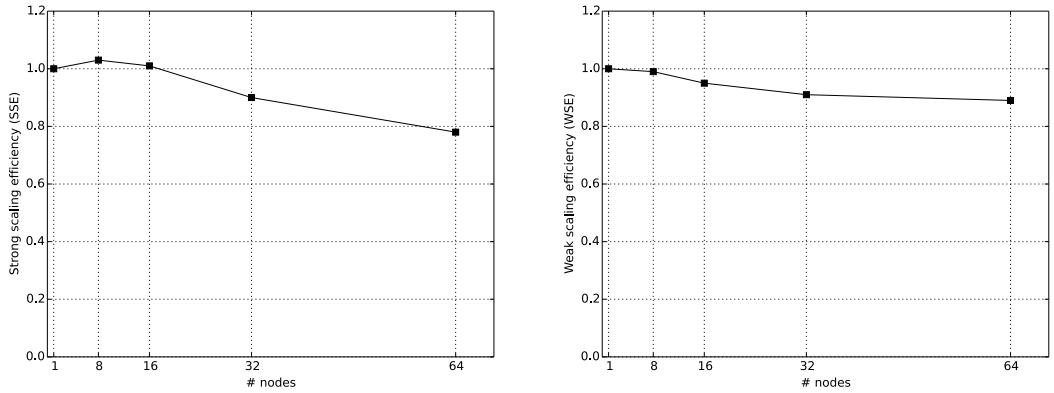
5 Conclusion

We analyzed HyperLoom performance using both synthetic and real test cases scaling up to hundreds of thousands of tasks distributed across hundreds of CPU cores. HyperLoom significantly outperformed Dask/Distributed in synthetic test cases ranging from $\approx 6.3\times$ to $\approx 2.2\times$ better performance for the *50kh* test case and from $\approx 7.6\times$ to $\approx 2.1\times$ better for the *gridcat* test case. We have also successfully deployed a pipeline to address the challenge of generating compound-target activity predictions for publicly available big chemogenomics datasets [3], which proves HyperLoom potential to be used for real-world end-to-end data processing applications.



(a) Strong scaling time. (b) Weak scaling time

Figure 4: HyperLoom scaling execution time (*mlchemo*).



(a) Strong scaling efficiency. (b) Weak scaling efficiency.

Figure 5: HyperLoom scaling efficiency (*mlchemo*).

References

[1] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[2] Red Hat. Red hat enterprise linux, 2017. [Online; accessed 31-March-2017].

[3] Jiangming Sun, Nina Jeliazkova, Vladimir Chupakin, Jose-Felipe Golib-Dzib, Ola Engkvist, Lars Carlsson, Jörg Wegner, Hugo Ceulemans, Ivan Georgiev, Vedrin Jeliazkov, Nikolay Kochev, Thomas J. Ashby, and Hongming Chen. ExCAPE-DB: an integrated large scale dataset facilitating Big Data analysis in chemogenomics. *Journal of Cheminformatics*, 9(1):17, dec 2017.

[4] Wikipedia. Infiniband — wikipedia, the free encyclopedia, 2017. [Online; accessed 31-March-2017].