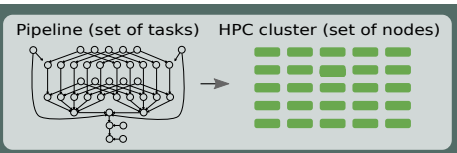
HyperLoom

A Platform for Defining and Executing Scientific Pipelines in Large-Scale Distributed Environments

S. Böhm[1], V. Cima[1], J. Martinovič[1], T. Ashby[2], T. Vander Aa[2], V. Chupakhin[3]
[1]IT4Innovations, VSB-TU Ostrava, Czechia; [2]IMEC, Belgium; [3]Janssen Pharmaceutica NV, Belgium

GOAL: Efficient end-to-end data processing in large-scale distributed environments made simple



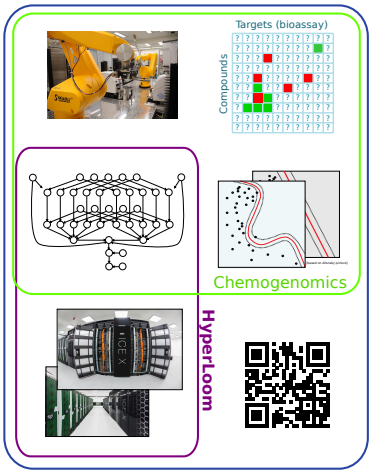
Motivation: Novel Drug Discovery

Chemogenomics

Modelling the interaction of the chemical compounds with biological organisms

Challenge: Running complex machine learning-based computational pipelines in HPC environments

Main objective: Easy to use framework for running large pipelines on HPC systems



Challenges

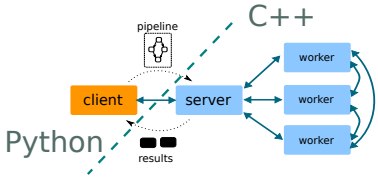
Pipeline size - pipeline can contain millions of tasks
Pipeline shape - pipeline can take shape of any directed acyclic graph
Pipeline heterogenity - pipeline can contain various types of tasks

Task execution time - task execution time varies from milliseconds to hours and is difficult to be predicted
Task output size - size of the output generated by a task is not known before task finishes

Cluster size - HPC clusters contain thousands of computatational nodes
Cluster heterogenity - computational nodes may provide different resources and their capacities

Architecture

client
• Python gateway to HyperLoom
• allows users to define **pipelines** and submit them to **server**



results
• output of submitted **tasks**

pipeline
• set of interconnected **tasks** represented as a DAG

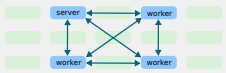
task
• basic computation abstraction (function, application)
• defines inputs, outputs, type and resource requirements

workers
• provide computational resources
• execute and run **tasks** as instructed by **server**

server
• central component
• handles **client** communication
• schedules tasks execution on resources available in **workers**

Workflow

STEP 1: Setup
• node allocation (e.g. PBS job)
• HyperLoom deployment

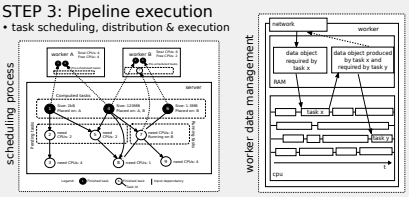


STEP 2: Pipeline definition
• pipeline definition and submission

```
import loom.client as lc

COUNT = 100
tasks = [lc.tasks.run("hostname") for i in range(COUNT)]
array = lc.tasks.merge(tasks)
client = client.Client("localhost", 9010)
future = client.submit_one(array)
#COMPUTATION
result = future.gather()
```

STEP 3: Pipeline execution
• task scheduling, distribution & execution



STEP 4: Result collection
• waiting for the submitted tasks
• collection of the results
• client connection termination

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 inter-node network traffic
• C++ core with a Python client enabling high performance through a simple API
• High scalability and native HPC support

Performance

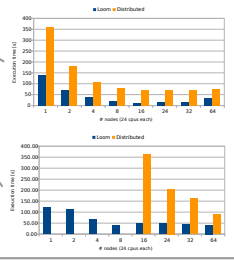
Test Scenarios

50kh
• test generating significant scheduling overhead
• homogenous pipeline
• ~50k short running independent tasks

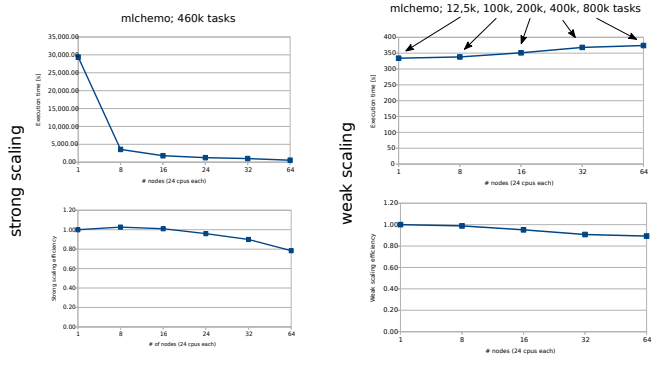
gridcat
• test generating significant network traffic
• ~5k tasks generating significant amount of data
• non-trivial pipeline shape

mlchemo
• test representing complex ML pipeline derived from a real chemogenomics use case
• highly varying task execution time
• non-trivial pipeline shape

Comparing HyperLoom to Dask/Distributed[1]



HyperLoom Scaling



[1]Rocklin, Matthew. "Dask: Parallel computation with blocked algorithms and task scheduling." Proceedings of the 14th Python in Science Conference. 2015.

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