Evaluation of Parallel Processing Frameworks with the GAWA Workflow

Daniel de Oliveira

Institute of Computing - UFF

Report presented to the Heising-Simons Foundation in fulfillment of the requirements for the scholarship - April 2022 - February 2023

Team

Raslan Ribeiro (IC/UFF)
Cristiano Singulani (LIneA)
Adriano Pieres (LIneA)
Luiz Alberto da Costa (LIneA)

Agenda

- Introduction
- A Brief Tour Through Gawa Workflow
- Evaluated Libraries: Dask and Parsl
- Adaptations in Gawa Workflow
- Performance Evaluation
- Conclusions and Roadmap

Motivation

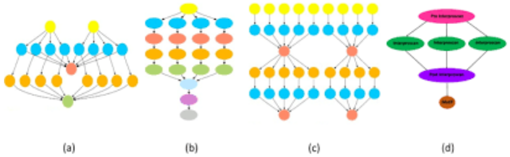
- The volume of data produced in academia and industry has reached unprecedented levels over the past decades.
  - This surge in data presents several challenges.
- Numerous scientific experiments rely on complex computer simulations.
  - Consume large datasets.
  - Require substantial computational resources to produce timely results.
- As experiments become increasingly complex, executing these simulations poses challenges, even in small- and medium-scale distributed and parallel environments like local clusters.
  - This is particularly evident in fields such as:
    - Bioinformatics.
    - Chemistry.
    - **Astronomy**.

Challenges - LSST Data Volume

- This scenario becomes even more complex when considering the LSST data volume.
  - It is expected that LSST will generate several terabytes (TB) of data per night, resulting in a final dataset of hundreds of petabytes over its 10-year project.
- Although LSST will have access to multiple dedicated HPC facilities, the ability to process data in real time will depend on the specific application.
  - *e.g.*, Santos Dumont Supercomputer @ LNCC
- The scale and frequency of LSST data will require **data processing frameworks**, libraries, and systems that **meet several performance requirements** within the LSST context.

Case Study: The Gawa Workflow

- Example: Gawa application (`https://github.com/linea-it/gawa.git`).
  - It is an adaptation of an application originally designed for detecting galaxy clusters in large surveys (specifically, the WaZP galaxy cluster sample of the Dark Energy Survey Year 1).
  - The application has been modified to detect stellar clusters, including faint halo clusters and dwarf galaxies.
- The Gawa application is compute-intensive and typically processes a large volume of data to identify stellar clusters in extensive optical surveys, such as LSST.
- It can be viewed as a scientific workflow, comprising a series of well-defined activities and their data dependencies.



1st Option - Workflow Management Systems

- Considering the Gawa application as a scientific workflow represents a step forward, but several challenges remain.
- One challenge is to determine the appropriate approach or system for modeling and executing the workflow.
  - Workflow Management Systems
    - Pegasus
    - SciCumulus
    - eScience Central
  - Entails a substantial learning curve.
  - Impose specific constraints on parallelization, which may hinder the execution of the Gawa workflow.
    - Lack of support for multiple levels of parallelization.
  - May require modification of portions or even the entire Gawa code.

2nd Option - Big Data Frameworks

- Big Data Frameworks, such as Apache Hadoop and Apache Spark, are widely used for scalable distributed processing of large datasets, typically ranging in the order of several terabytes.
- However, using these frameworks requires rewriting existing applications to adhere to a rigid program structure.
- Developers have limited control over data locality in these frameworks.
- Data is randomized across multiple storage nodes, leading to unpredictable data communication patterns.
- This lack of control over data locality makes it challenging to explicitly minimize data movement, which can be particularly concerning for applications like those used in LSST.

3^{rd} Option - Parallelization Libraries

- The advantage of this approach is that programmers are free to use a scripting language they are familiar with.
- The major drawback of script languages is their scalability.
  - Such general-use libraries are specifically designed to enhance script scalability in a relatively straightforward manner.
- Although each approach has distinct advantages and disadvantages, this option appears to be the most suitable for the Gawa workflow and other existing applications within the context of LSST.

Goals

- The goals of this work are as follows:
  1. Analyze the original Gawa source code and identify its main activities, particularly those that are compute-intensive and associated with performance bottlenecks.
  2. Adapt and execute the Gawa application using Dask, a Python library for parallel computing.
  3. Adapt and execute the Gawa application using Parsl, a flexible and scalable parallel programming library for Python.

A Brief Tour Through Gawa Workflow

- Gawa is a workflow implemented in Python that aims at identifying stellar clusters using wavelet transformation in a catalog of stars filtered by isochronal masks.
- Originally used to detect a cluster of galaxies (WAZP), but not scalable to LSST data volume.
- To apply Gawa in areas larger than those available in LSST Data Preview simulations, and also with star clusters included, the LIneA team developed a code to create simulated star clusters.
- The workflow of the simulations is well-defined and composed of 12 activities

Gawa Simulation Workflow

1. Load user-defined parameters in JSON file;
2. Download input data files (isochron file);
3. Create the footprint map;
4. Load the files with the photometry of the astronomical survey;
5. Correct the star photometry by extinction;
6. Scale of sampled fields;
7. Create the file with information about the simulated clusters;
8. Simulate clusters;
9. Merge the simulated clusters catalogs with the field stars catalogs;
10. Remove stars in dense fields;
11. Recount the stars in the simulated clusters and calculate the absolute magnitude of the clusters;
12. Generate as output figures presenting the distribution of clusters, their characteristics, list of simulated clusters, *etc*.

* All generated files are in FITS or ASCII format.

Gawa Detection

- Most of the Gawa workflow is implemented in Python
- It uses the Multi-Resolution Filter (`mr_filter`) - a legacy code developed in C that is invoked from the main Python code.
- Although C code often provides better performance, in this context, this characteristic may hinder the full utilization of parallelization libraries by the Gawa workflow.
- In the initial tests, we used data from the Dark Energy Survey (DES - `https://www.darkenergysurvey.org/`) for cluster detection. Subsequently, we incorporated LSST Data Preview 0 (DP0 - `https://dp0-1.lsst.io/`) data into our experiments.

Dask - https://www.dask.org

- Dask is a library that aims at providing parallel capabilities for applications developed in Python.
- Composed of two main modules:
  - data collection
  - a task scheduler
- Provides multiple distributed data structures with APIs similar to the PyData ecosystem
  - DataFrames[1]
  - Bags[2]
  - Arrays[3]
- Each collection operates in parallel on existing datasets, and these datasets are not required to fit in memory (this is a prerequisite for other libraries and frameworks).

[1] https://docs.dask.org/en/stable/dataframe.html
[2] https://docs.dask.org/en/stable/bag.html
[3] https://docs.dask.org/en/stable/array.html

Example - Dask Dataframes

```
1  import dask.dataframe as dd
2  df = dd.read_csv('2014-*.csv')
3  df.head()
4      x  y
5  0  1  a
6  1  2  b
7  2  3  c
8  3  4  a
9  4  5  b
10 5  6  c
11
12 df2 = df[df.y == 'a'].x + 1
13 df2.compute()
14 0    2
15 3    5
16 Name: x, dtype: int64
```

- Besides the obvious differences (*e.g.*, `IMPORT` in line 1), the main difference is that in Dask DataFrame, the user triggers processing by calling the `.compute()` method (line 13).
- Dask DataFrames are recommended when datasets do not fit in main memory and the application has long processing tasks

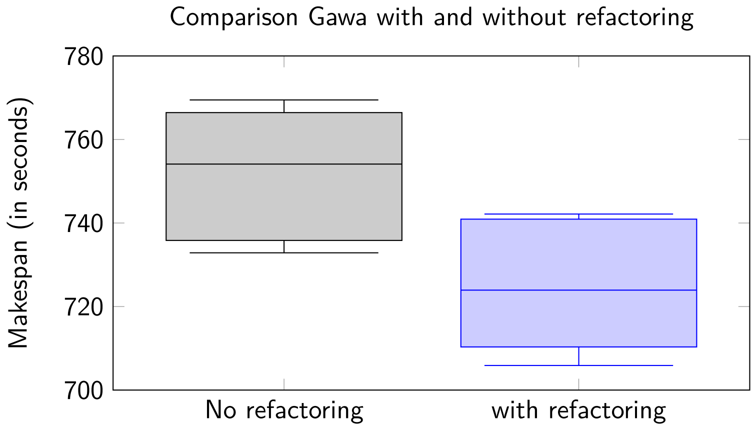
Parsl - https://parsl-project.org/

- Parsl is also a library for the parallelization of Python programs.
- Allow the developer to annotate the original Python code where Parsl can execute in parallel
- This annotation (called *decorator*) is performed using *Annotated Functions* (apps)
- Identified Apps execute concurrently while respecting the data dependencies of the original script.

```python
@python_app
def generate(limit,delay):
    from random import randint
    import time
    time.sleep(delay)
    return randint(1,limit)

rand_nums = []
for i in range(5):
    rand_nums.append(generate(10,i))

outputs = [i.result() for i in rand_nums]

print(outputs)
```

Library-independent Adaptations

- Replacing `FOR` loops by `MAPS`.
- Freeing unused variables
- Avoiding Comparisons with Zero



Refactoring presented performance improvements up to 4.06% in an Intel® Core™ i7-10750H CPU @ 2.60GHz × 12, 32GB RAM, 500GB disk SSD consuming data from DES

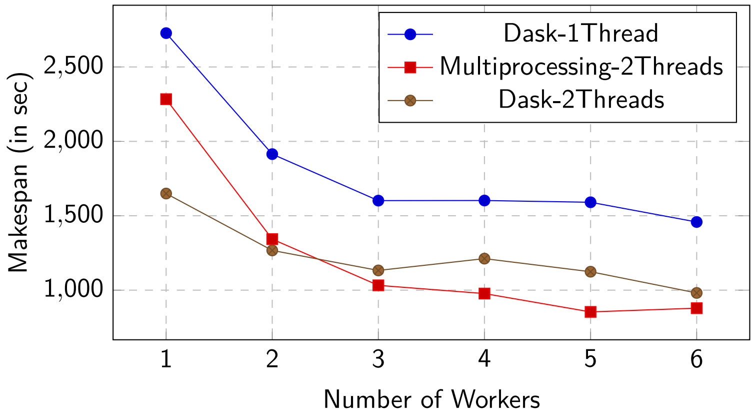
Adaptations for using Dask Data Collections

- Arrays to Dask Arrays.
- Choosing the proper `chunk` size is not a trivial task in Dask.
- Array chunks cannot be too big (otherwise the application will run out of memory), or too small (the overhead introduced by Dask becomes overwhelming).
- Usage of `to_dask_array()` to avoid informing the chunk size of the new array.
- Although this modification is not too complex to be done for a single array, it is necessary to convert **ALL** NumPy arrays in Gawa.

Adaptations for using Dask Data Collections

- The original Gawa code was programmed using `multiprocessing` package of Python.
- It provides the `Pool` object which offers a way to parallelize the execution of a function by distributing the input data across multiple processes (*i.e.*, data parallelism).
- Using Dask, the developer has to start a `Client` to use the `futures` interface
- The programmer has also to define important parameters for the parallel execution at this point such as the number of workers (`n_workers`), the maximum amount of memory that can be used (`memory_limit`) and the number of threads per worker (`threads_per_worker`).

Comparing Dask and Gawa Multiprocessing



Executed @ ICEX cluster at LIneA (HTCondor) consuming data from DES

The performance of Dask is similar to the multiprocessing. More analyses with larger executions (both in terms of input data and the number of workers) would confirm if Dask could outperform the multiprocessing version of Gawa (and Parsl)

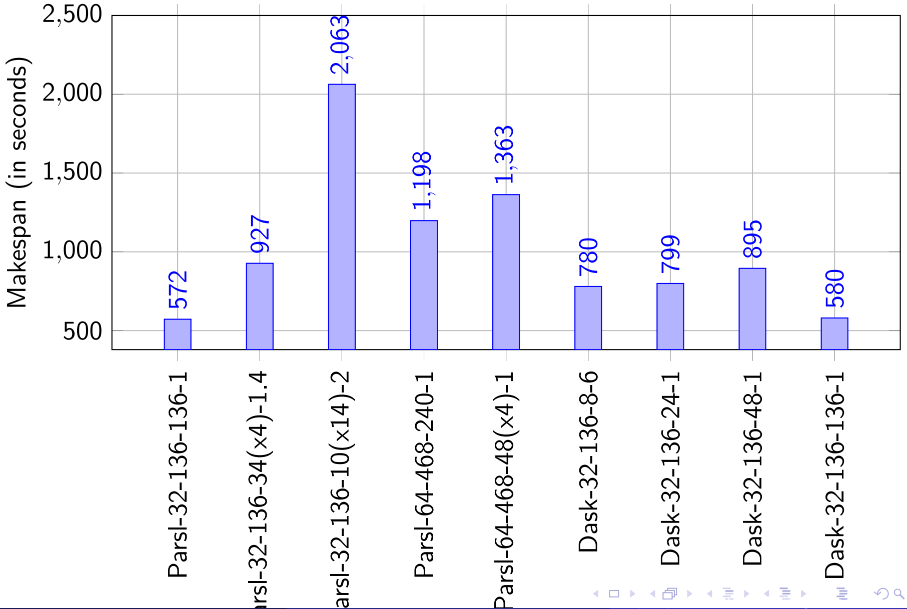
Adaptations for Parsl

- Parsl does not require to use of specialized data collections or specific task scheduling mechanisms.
- Usage of *decorators*.
- To avoid using parallel strategies in portions of the code that are not computing- or data-intensive, only a few functions were chosen to be annotated and parallelized
  - `create_mosaic_footprint_job`
  - `compute_cmd_masks_job`
  - `run_gawa_tile_job`
- Each one of these functions represents a `@python_app` for Parsl.

Comparing Dask and Parsl

- Executed the Gawa workflow using both Dask and Parsl in the Santos Dumont (SDumont) supercomputer @ LNCC
- The SDumont has a total of 36,472 CPU cores, distributed in 1,134 heterogeneous computing nodes
- We have reserved 2 x CPU Intel Xeon E5-2695v2 Ivy Bridge, 2,4GHZ, and 2x Intel Xeon Cascade Lake Gold 625 for the exe- cutions
- We have executed Gawa consuming data from DP0 with 5,000 square degrees
- Each execution of the workflow varied the following parameters
  - $NSide = \{32, 64\}$
  - $Tiles = \{136, 468\}$
  - Number of workers
  - Number of CPUs per worker
- The label of each execution is in the form "*Library-Nside-Tiles-Workers-CPU_per_worker*".

Comparing Dask and Parsl



Conclusions and Roadmap

- We have explored two different HPC libraries (Dask and Parsl) and environments (local cluster and SDumont supercomputer)
- Dask requires a series of code adaptations to execute properly.
    - recommended for projects that are being developed from scratch
    - no legacy code!
- Dask requires a parameter fine-tuning step
    - Can be challenging in scenarios where the same workflow can be executed in different federated environments (which is the case of LSST)
- Only functions that consume large amounts of data are recommended to be parallelized using Dask.
- Parsl is the recommended library since it requires less modifications in the original code.
- Challenge: the use of third-party software within the Gawa code.
    - The Multi-Resolution Filter (`mr_filter`) is a legacy code developed in C which prevented Gawa workflow in both Dask and Parsl from fully benefiting from the parallelization capabilities.

Thank you for your attention!