pyProCT Optimization

by Pol Alvarez

Supervisor: Víctor Gil Sepúlveda

GEP Tutor: Jasmina Berbegal

Specialization: Computation

Submitted as Degree Final Project
of Bachelor Degree in Informatics Engineering 22/02/2015

Contents

1. Introduction

1.1 Problem Justification

Python Protein Clustering Tool (pyProCT from here onwards) Software is an open source software developed by Victor Gil Sepúlveda for cluster analysis (see Github repository). This software provides an improved clustering performance through the initial definition of an hypothesis or goal. First it computes the distance's matrix of each pair of elements; it then uses a number of different cluster analysis algorithms on the dataset trying to estimate the best parameters for each one, and, finally, it rates the performance of each method and parametrization with a common scoring function.

The goal of this project is to refactor pyProCT using the COMPSs programming model and framework developed by the Barcelona Supercomputing Center (BSC). This framework provides a sequential programming interface to achieve a parallel and optimized execution pipeline. Programming models are valued by it's capabilities and limitations so in order to explore, contribute and help to further develop the COMPSs framework it is mandatory to use it on real projects to test and debug it.

COMPSs provides a good parallel execution whilst allowing the developer to code in a sequential-oriented way. This differs from other models and paradigms which require the developer to have a deep knowledge of the hardware executing the code such as MPI interfaces and OpenMP C pragmas amongst others. And this project tries to explore and help to develop it by using in on existing to software in a real environment such as pyProCT.

1.2 Objectives

I will refactor pyProCT with COMP superscalar for python [1] (pyCOMPSs from now onwards). The objective is to see if it reduces the execution time on cluster and grids and/or the complexity, thanks to the easy-to-use sequential programming paradigm. To achieve this the work will be three-folded and incremental following the next pattern:

1. **Program Analysis** of the actual performance. This will be useful for two reasons. On one hand we need to analyse the program to, after improving it, measure the performance optimization. On the other, a deep analysis will help to identify the algorithm's bottlenecks and slowest parts as well as finding which are the best optimizations to perform.

2. **Optimization** of the algorithm. Because each clustering execution is unrelated to others this step is embarrassingly parallel and so, even prior to the analysis part, some sort of parallelization will surely speed up the algorithm. pyCOMPSs programming model has been chosen because it aims, not only to exploit the inherent parallelism of the program but also, to ease the development of distributed infrastructures in which the software will run.

3. **Optimization's analysis**. Finally we will analyse, compare and evaluate the performance improvements with respect to the initial code. It is important to note that this step differs from the first one on many aspects. Here the focus will be to analyse the changes introduced by the last modification and how it affects the performance. On the first section the focus will be to plan the new modifications, check how they behave with the other ones; merging them together if there is a performance improvement or deciding if it's worth to further investigate how to integrate them with the rest with a performance improvement or discarding them otherwise.

[1] More info on COMPSs documentation on Section 6

1.3 Scope

The analysis and optimization of this software will follow a computational and statistical approach. Following this approach comes the first mentioned optimization with COMPSs. The other optimizations will be decided after the initial analysis of each cycle (see 4. Development Method). However the scope of this optimizations will be limited. No architecture-specific analysis or improvements will be done. All possible modifications will fall into one of the following categories:

1. **Distances matrix** computation improvement. The initial calculation of the distances can be space-optimized. A deep analysis will tell if it's worth implementing a more space-efficient function.

2. **Task level**. Considering each clustering analysis a task, we can improve the performance of specific algorithms. This includes improvement of the cluster analysis algorithms thanks to either parallelization or different and faster implementations. It is important to note that a task level optimization might not result in an overall performance improvement due to, in example, bottlenecks and execution order. Because of this, the initial analysis is necessary to guide possible improvements. Also the COMPSs refactoring is going to be the first modification because having a robust, prioritizable parallelization will allow us to focus on the tasks making sure that the changes do improve, not just the task performance but, the overall execution and helping sidestep possible bottlenecks. Finally, clarify that this category contains both the optimization of the clustering algorithms as well as the optimization of the best parameters estimation, which may come together or not.

3. **Scheduler level**. In this category we find the optimizations that will speed up the software overall performance without changing the specific algorithms. COMPSs refactoring belongs here as well as priority executions, to avoid bottlenecks and parallelizations. Other possibilities are clustering algorithms execution pruning based on other's best scores or pipeline modifications.

1.4 Methodology

Due to the fact that there are many possible optimizations of pyProCT (some of them complex enough to be a full project), the limited time and the incremental development of the improvements, we have decided to use an Scrum based methodology. We will set time-variable cycles at the end of which the work will be evaluated.

I am going to use Paraver and Extrae, for traces' analysis; pyProCT-regression, to validate the new implementation. These tools, as well as pyCOMPSs and pyProCT, are still on development and not fully tested. On this scenario the Scrum methodology is the best. Having cycles means that it's easier to evaluate if some trials lead to a dead-end, are best implemented on another way or, simply, they are not feasible because they are not supported.

Each cycle will be bound to an specific modification. To begin each cycle we will analyse the current state of the project and how previous work affects the code to define which is the next goal and the results we expect to see. Once decided the work plan, we will proceed to it's implementation.

Once finished we will check if the goals where achieved. It is important to note that even if the cycles duration will be variable, because the complexity of each optimization can vary a lot, it will be soft-decided at the beginning. This will help to keep track of the work easily, decide if a particular modification is taking too long and it will reflect the possibility that an optimization does not improve the overall performance, case in which the results will be analysed and reported nonetheless prior to planning the next work to be done.

This is the most effective way because performing a full initial analysis and deciding at once all the optimizations to implement does not take into account how one modification might affect the next one. We also don't know if the proposed methods are really feasible.

The validation on this project will ensure the correctness of the refactored code. In order to do so different testing methods are going to be proposed. On one hand we will implement a black-box testing. To do so we will gather a large enough and significant number of data sets.

Then the original software will be run on those, storing the results of the executions. This will be our first reference point. Then we will implement an script to compare the original results with the ones provided by modified software. For each big iteration these data sets will serve to test the correctness of the new code. However, we will be working with large data sets and this kind of testing can be too time-consuming to be performed as often as desired.

The code is already part of a github repository. I will fork the code in order to track the changes. For each cycle a new Github milestone will be opened to organise and gather together all the issues (opened on Github too) and changes affecting the same optimization/cycle.

To work on this project a laptop with the text editor Sublime Text 3. It will also use and have installed all the required software to run the code on the Mare Nostrum machine (through ssh), fork and manage the code versions with git, run the tests and instrument the code (for further details see both 8.2 Hardware Resources and 8.3 Software Resources sections).

1.5 Limitations and Risks

As stated earlier, most of the used tools and software is still under development so many problems are to be expected. To deal with it I proposed the Scrum methodology (which thanks to it's incremental nature will help to discard unfeasible modifications when finding unsupported features on the tools) and I started to work at the BSC's COMPSs group in order to know more about the internals and to be able to ask questions faster.

Additionally the parallelization could introduce the problem of the reproducibility on the testing.

The reproducibility problem, defined as the impossibility to repeat an exact execution of the algorithm because of some stochastic parts, such as random initial parameters estimation for example, could difficult the validation and testing part. This could lead to a number of problems. First, the inability to use the black-box validation if two executions with the same data set lead to different results. This clearly affects all the parts of the process involving some

kind of randomness. To control this, in case it does affect the testing process, we will try to eliminate the stochastic issues with things like random seeds and manual and fixed parameter estimation determined on the unit tests.

Another issue could be the time. To mitigate this problem the initial set-up phase before the SCRUM iterations has been added (see subsection 7.3.4 on Tasks Description). The goal of this is to automate the analysis, execution and all the other time-consuming tasks not related to the actual development of the optimizations.

More problems such as the inability to correctly execute the on the Mare Nostrum III server will be addressed by counting on the BSC team and the project director. The usage of the extrae could also prove to be difficult, to this end the help of an extrae expert, who has already been contacted, is taken into account as an exterior consultant.

1.6 Context

Nowadays the amount of digital information available is exponentially increasing. Just on 2015 we generated almost 8.000 exabytes of information. Facebook generates 105 terabytes of data each half hour, more than 48 hours of video per minute are uploaded to youtube and google has at least 1 million queries/minute. But why do we observe this massive increase? To start the cost of creating, managing and storing information has dramatically dropped: EMC Corporation estimates that on 2011 this cost has been cut to a 1/6 of what it was on 2005. But more importantly people is more connected than it has ever been; mobiles, websites and social channels are just some examples of a whole new world of data-generating people interactions.

In this scenario is where we found the hot topic of today: Big Data. So what is it?, usually the term is used referring to data sets too big or complex to be processed with traditional data applications or on-hand management tools. According to the IT giant Gartner, Inc Big Data can be characterized by the "3 Vs", velocity, volume and variety [**?**]:

> "Big data" is high-volume, -velocity and -variety information assets that demand

cost-effective, innovative forms of information processing for enhanced insight and decision making.

However all this raw information needs to be processed and categorized in an effective way before being used. Cluster analysis methods are one of the most used tools to address this issue.

The term **cluster analysis** (first used by Trion, 1939) refers to the task of sorting similar objects of a data set into groups (called clusters) in a way that the degree of similarity between each pair is maximal if they belong to the same cluster and minimal otherwise. Data sets can be imagined as points in a multidimensional space, where each feature of an object would represent a dimension. The CA methods need to identify, as efficiently as possible, the denser areas and group the into clusters.

Thanks to the clustering we can reduce the size of large data sets by extracting the most relevant information, usually the common features of a group or a subset of representatives. Cluster analysis (CA from now onwards) techniques thrive in the Big Data world because it's not feasible to manually label objects, there is no prior knowledge of the number and nature of the clusters and, also, their identifying traits may change over time.

It is important to note that cluster analysis it's not an specific algorithm but rather the general task to perform. Due to the fact that the similarity criteria it's subjective and can change a lot between data sets, there isn't an optimal clustering algorithm. This is the reason why there are so many clustering algorithms, each with it's advantages and inconveniences. Each algorithm uses it's own kind of cluster model that defines how the algorithm groups the items and defines the clusters. Some of the most relevant examples are:

Hierarchical Clustering Analysis (HCA)

These methods seek, as their name indicates, to build a hierarchy of clusters. These can be done by starting with all elements in one clusters and the divide them in a "top down" way, this method is called **Divisive**. Opposed to this one we find the **Agglomerative**

method, where each data point starts in a different cluster merging them as one moves up the hierarchy.

Centroid Clustering,

On these algorithms the similarity between different clusters is defined as the similarity between their centroids. **K-means** clustering is one of such methods, on it, each observation belongs to the nearest centroid which, in turn, serves as the representative or prototype of the cluster.

Distribution-based Clustering

Clusters are modelled by statistical distributions. On this category falls the well-known **expectation-maximization (EM) algorithm** which uses multivariate normal distributions.

Density Clustering,

These methods follow the intuitive notion, described earlier, of considering the observations as clouds of points in a multidimensional space and so, they identify clusters as connected dense regions in the data space. **DBSCAN** is one of such algorithms and it's both one of the most common as and most cited in scientific literature.

It is also possible to classify clustering methods by some other properties such as:

Hard Clustering,

where each element belongs to a cluster or not.

Soft Clustering,

where each element has likelihood of belonging to a certain cluster.

Cluster analysis methods can be applied to a wide range of subjects. Basically it can be used in any context where finding groups in sets of data is useful, for example:

Image segmentation,

dividing an image into clusters or, more appropriate on this case, regions enhances a

number of computer vision methods. Some examples are border detection or object recognition. [?]

Market analysis,

grouping enterprises [?] or consumers [?] to perform better market analysis or custom ads for each kind of consumer.

Education tracking,

grouping students to keep track of their record and apply more custom techniques to each student needs. [?]

Mathematical chemistry,

to analyse, group and find structural similarities in chemistry compounds, minerals, and any kind of material for which a chemical analysis is convenient. [?]

1.7 Stakeholders

The implied stakeholders here are, primarily the COMPSs developer team and, second, the rest of the research teams implied on GRID and cluster execution platforms interested on Cluster Analysis. They are the ones who will mostly benefit from this project. As stated earlier, this project wants to develop and explore the COMPSs framework. With this the programming model will be refined and, thanks to it's usage, more desired features can be found. Also the future usage of pyProCT will help to "advertise" and enhance it's possible diffusion, which is the main goal to keep this kind of programming models and frameworks, not just alive, but on a develop and improvement route.

On the other hand the pyProCT software was originally intended for protein clustering. However, the program core is quite generic. Thanks to the implementation of new plug-ins and modules to load, transform and use other kind of data inputs, the software is expected to suit a large group of researchers without too much work. Providing support for other usages is not part of this project but a possible speed up and performance enhancement will help them all

and so, they are considered the main clients of the software itself and they will benefit from it.

1.8 State of the Art

Most of the clustering analysis methods are not new, however with the dramatical increase in data size mentioned earlier, researchers have focused on improving their performance as much as possible. From this need arise new, but more rough, methods such as **canonpy clustering** [?] which can process huge amounts of information but it just a pre-partitions data to then analyse smaller partitions with slower methods.

The increasing amount of information each data point contains it's also a problem for some algorithms. This information leads to high-dimensional data which, in turn, causes problems to a big part of the modern algorithms. This is known as the curse of dimensionality, which basically points out the fact that high-dimensional data often becomes sparse due to the large volume of space. It is important to note that this problem is not due to data itself but to the algorithm used. Some modern approaches try to overcome this difficulty by reducing the data-dimensionality, with methods such as **principal component analysis** [?], use just some part of it, like in **subspace clustering** [?] which have adopted ideas from density-based algorithms.

Apart from the clustering analysis techniques the focus of the project is the COMPSs refactoring. The election of the framework has been made according to two reasons: one, the proximity of the BSC research team which will ease the development, two, and most important, the framework is aimed to distributed computing on which the CA tasks are best executed.

1.9 Document Structure and Notes

This section will give an overall view of how the document is structured. I decided to give this document sort of a chronological order because of two reasons: first, following along the work I did makes the whole document easier to understand and second, the work has been incremental,

each iteration relying upon the previous decisions, so it seems reasonable to report the results and problems in the order they appeared. l

The structure will be similar to the project methodology described on 7. Temporal Planning section. First I will describe how pyProCT works which matches the Code Familiarization task.

The two next goals were to find which tools to use and to set up the workspace to start de Scrum iterations. To do so I wrote an small program in order to test many things and find possible solutions on a much smaller scale than pyProCT. It helped me to see and learn how to deal with reproducibility problems amongst other validation issues, understand how th current pyProCT's scheduler works and finally to test and learn how to use the analysis tools.

The Tic-Tac-Toe section describes all the scheduler and analysis tools information. The next step, prior to start working on the refactor, was to set up a reliable validation method to ensure the correctness of the new versions. pyProcT-Regression is the developed software used for black-box validation; on the Regression section I will describe all the information related with validation (because I deemed excessive and useless to develop a full testing software for the tic-tac-toe game).

Finally the refactor section contains the work related with the integration of a pyCOMPSs scheduler for pyProCT.

As a final remark I want to say that all tools used have documentation. However being in development some of them are outdated. Wherever the documentation of the tool is good enough I just described the features or issues directly related with this project.

Of the analysis tools I just explained how I instrumented the code, together with the found issues, and how I obtained the desired visualizations and data.

For pyProCT I described the execution flow (which requires in-depth code knowledge) and summarized and updated the control script section. The actual description of the algorithms and parameter estimation are complex topics; however, their documentation is good enough so I decided just to list them and refer the reader to the original documentation for more information.

Similarly for MareNostrum III environment I mention the creation of bash scripts (to speed up the work, easily submit different versions of pyProCT, different schedulers, with or without tracing and diverse parametrizations) configuration files and usage but I do not describe them unless they are related to issues.

2. Methods

2.1 Software design description: pyProCT

pyProCT is a clustering analysis software. As we stated earlier, correct usage of clustering analysis methods is not easy: right algorithm selection, better parameters estimation or appropriate result analysis are just some of the problems that CA tools user faces. However, we also showed that CA is present in a lot of different subjects and is used, or would be useful, to people with limited knowledge both on algorithmic methods and statistics. On the other way around we also find that CA specialists may not be able to correctly assess the results of a clustering due to the nature of the data itself.

This clustering tool aims to reduce this gap by using five different algorithms and allowing the user to define a clustering goal or hypothesis. With this and some more options, most related to the parameter estimation of each method, it tries to find the best algorithm and it's parameters for the inputted data. This way, through a more "semantic" approach we expect to guide pyProCT to find the combination that will produce the best clustering, without forcing the user to deeply understand the pros and cons each method w.r.t to an specific kind of data.

To schedule the execution of the algorithms uses **pyScheduler** controller. It features three modes: sequential, parallel, using python's multiprocessing module, and parallel using MPI. The refactor will add fourth mode to run the tool with pyCOMPSs. It is important to note that the modifications will be limited to pyProCT, so COMPSs will act as a substitute of the current controller, not as a new scheduling method inside pyScheduler.

2.1.1 Algorithms

pyProCT uses the following five algorithms to find the best clustering. It also features a last one which clusters the data randomly. This one is used for comparative purposes so it won't have more consideration that the utility it provides for other's behaviour analysis.

1. K-medoids
2. Hierarchical
3. DBSCAN
4. GROMOS
5. Spectral

For more information about the actual implementation and parameter pyProCT dropbox documentation[1]

2.1.2 Execution Flow

The execution flow of pyProCT can be subdivided into four main sections linked to the JSON script structure:

Global,

initialization of the software by reading parameters and options, parsing the JSON script, setting up the workspace and create the scheduler to be used.

Data,

construction of the distance matrix to be used. It offers three options: load, distance and rmsd.

[1] https://dl.dropboxusercontent.com/u/58918851/pyProCT-SupportingInformation.pdf

Clustering,

calculation and evaluation of the clusterings.

Postprocess,

processing of the results to offer useful information about the clustering found.

On the next sections each part is going to be described, both it's execution flow and the json parameters associated with it.

Global

The global section is the responsible of initializing everything for the execution. This part is located on the main.py file and ends up calling the corresponding driver with the correct parameters. This is the main.py structure:

This main file creates and initializes the Driver class which is the one orchestrating all the execution. Then the Driver class creates the workspace handler and saves the parameters before starting the Data, Clustering and Postprocess sections. The following figure shows encased in red the part corresponding to the global section inside the Driver. The next sections will expand the boxes Data, Clustering and Postprocess.

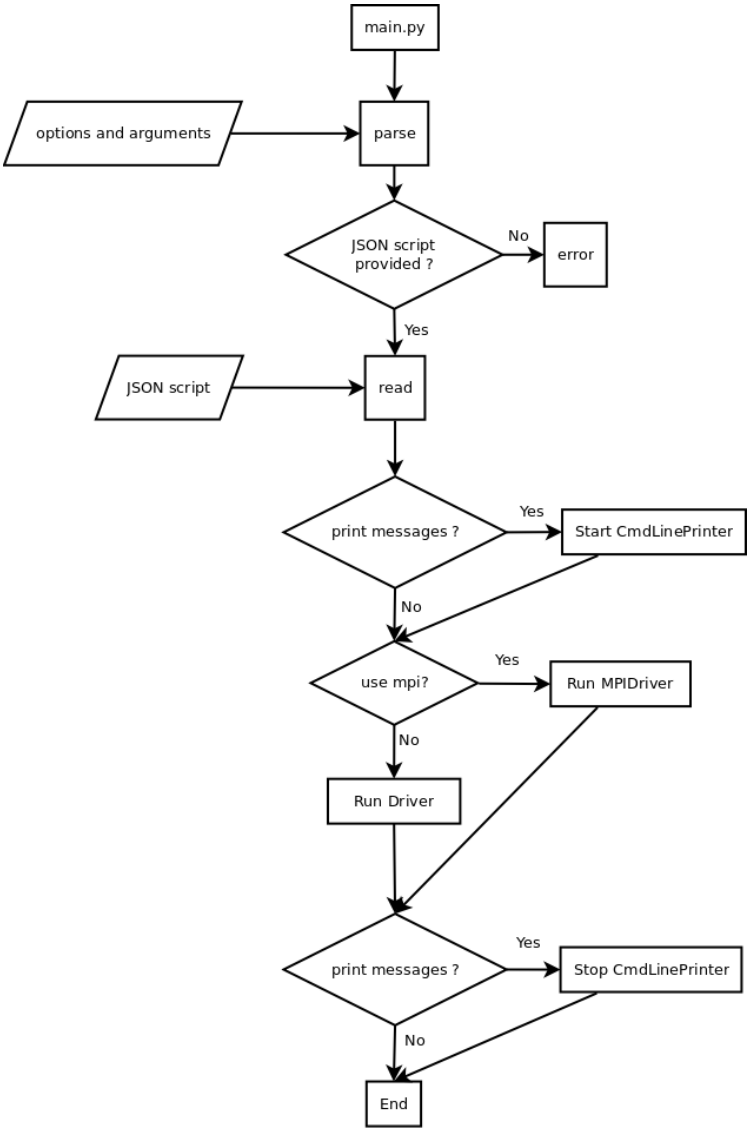

Figure 2.1: Global Section Execution Flow

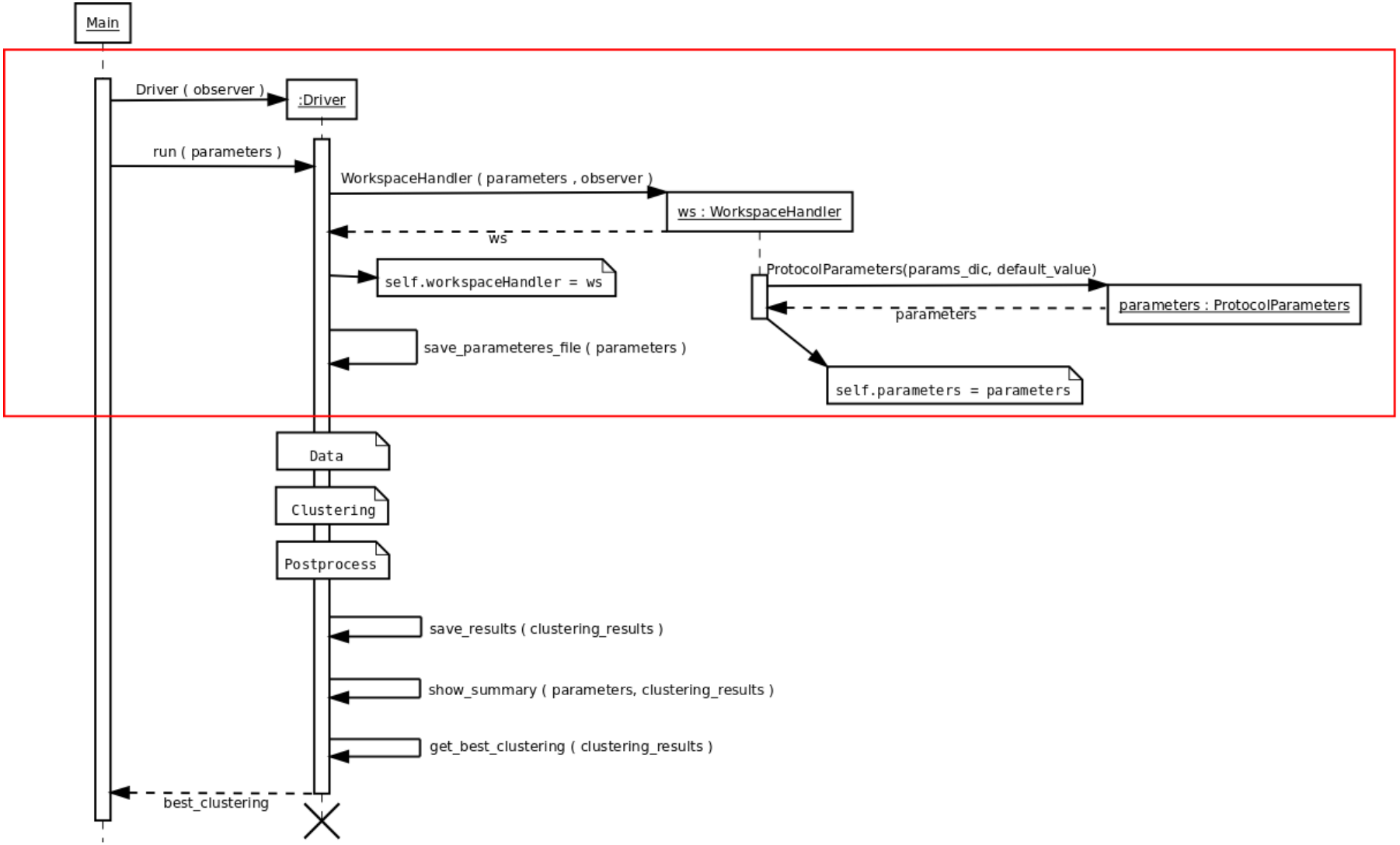

Figure 2.2: Global Section Execution Flow

The global section parameters that can be specified on the json file are divided into two groups: Control and Workspace.

- **Control**

  Scheduler type, defines the kind of scheduler to use (serial, parallel, MPI or, after the refactor, pyCOMPSs)

  Number of processes, if the parallel scheduler type is selected this option defines the number of processes to be used.

- **Workspace**

  Base, is mandatory and defines the base workspace path.

  Tmp, defines the folder to store temporal files.

  Matrix, defines the folder to store the distance matrix (if applicable).

  Clusterings, defines where cluster-related files are going to be stored, however the clusterings are stored as part of the results file.

  Results, defines where the results file should be stored.

  Parameters:

  Overwrite, if true, existing folders will be removed before execution.

  Clear after execution, defines the folders to be removed after execution.

Data

This section defines how the distance matrix should be build. Essentially it runs the DataDriver class' function run() with the WorkspaceHandler initialized on the Global section and the retrieved parameters. This DataDriver then initializes and returns to the driver the DataHandler and MatrixHandler to be used later. The first one is directly instantiated with the corresponding parameters. The second one, on the other hand, first loads the matrix calculator defined

on matrix::method of the json control file. With this calculator, the data handler and the parameters, it computes and returns the desired matrix handler.

The following figure shows a simplified sequence diagram of this process:

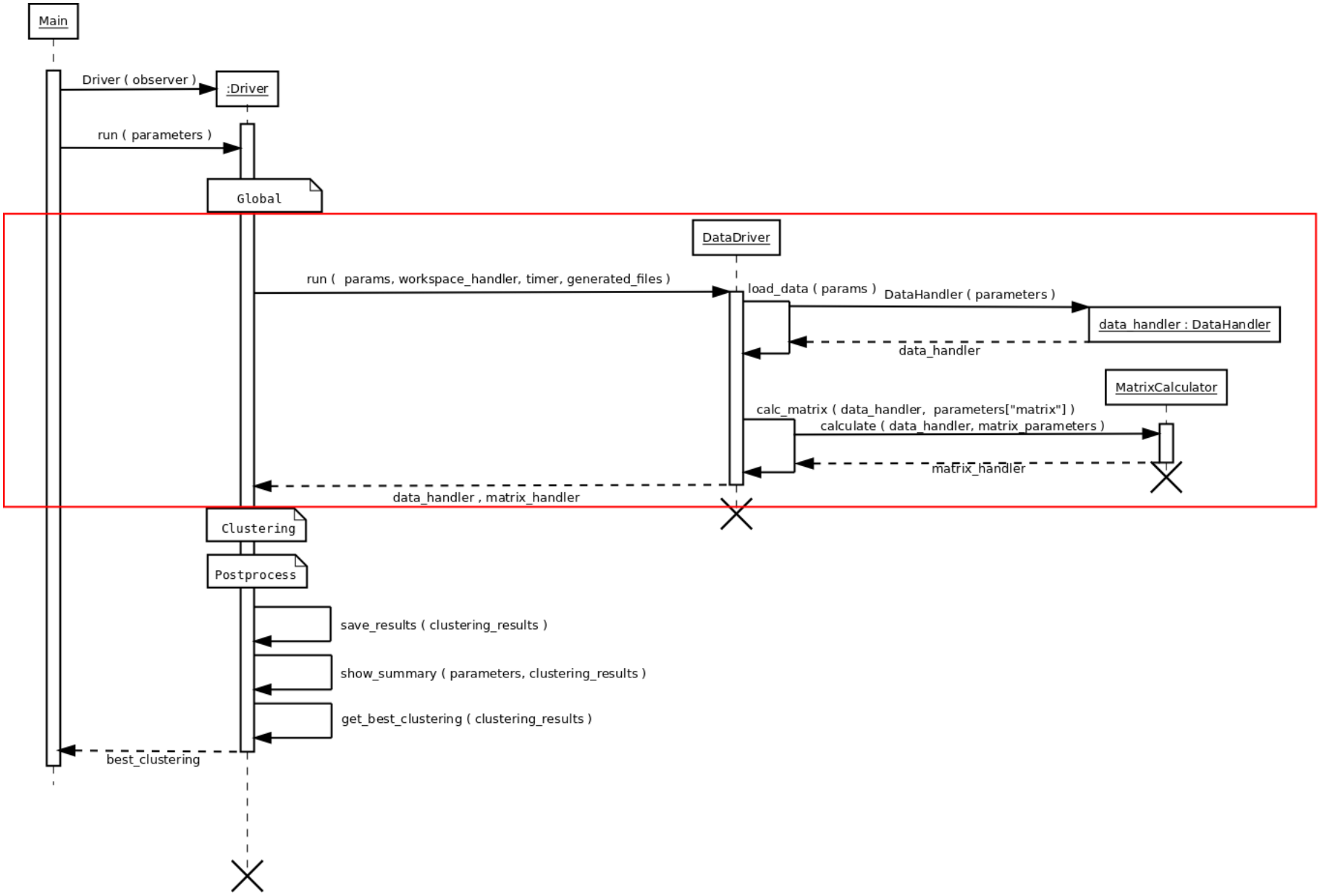

Figure 2.3: Data Section Execution Flow

The data section parameters specify the type and origin of the data, the method used to calculate the distance matrix and some more matrix-related parameters:

Type, sets the kind of data loader to use for the dataset.

Files, defines the location of the input files.

Matrix

Method, selects the method used to calculate the distance matrix.

Parameters, allows to customize some parameters used to by the distance matrix calculator like:

Calculator Type, must be one of the local pyRMSD installation available ones.

Fit Selection, for distance or rmsd methods.

Body Selection, for distance method.

Calculate Selection, for rmsd method.

Path, in case we are loading the matrix.

Image, setting this section will result in rendering a visual representation of the matrix.

Filename, desired path of the rendered image.

Dimension, sets the leading dimension of the matrix image [default:1000px]

Filename, name of the file where the distance matrix will be saved (if applicable) inside the folder defined on workspace::matrix section.

Clustering

This section is the one performing the actual clustering exploration and evaluation of the results.

As Figure 2.4 shows, the driver calls it's clustering_section() function which checks wether it needs to perform the exploration or load an existing clustering.

If the method selected is "load" then the function from_dic(...) turns the data into a Clustering instance.

If "generate" is the selected method then it calls perform_clustering_exploration(...) which initializes and runs the ClusteringProtocol class. This one, in turn, runs and initializes the classes ClusteringExplorer, ClusteringFilter, AnalysisRunner and the BestClusteringSelector.

This ClusteringExplorer deals with the actual exploration pipeline. It generates diverse parameter structures for each defined CA algorithm and adds them to the scheduler tasks queue, runs the scheduler and returns the resulting clustering_info structures.

The ClusteringFilter tries to reduce the size of the clustering. To achieve this it eliminates the clusters whose parameters are outside the defined ranges on the evaluation section, removes the not selected clusters and checks that there are no repeated clusterings amongst the selected ones.

The AnalysisRunner is the one handling the evaluation of the selected clusterings. Similarly to ClusteringProtocol, it creates an scheduler instance, queues the parametrization of each clustering analysis into it and runs it. Finally, it attaches the results to the clustering structure evaluated.

The BestClusteringSelector normalizes the calculated evaluations, scores each evaluation with the defined criteria and finally returns the id with higher score and the score itself.

Finally the ClusteringProtocol returns the clustering_results containing: best_clustering_id, selected_clusterings, not_selected_clusterings all_scores. This results are then returned to the Driver for the postprocessing section.

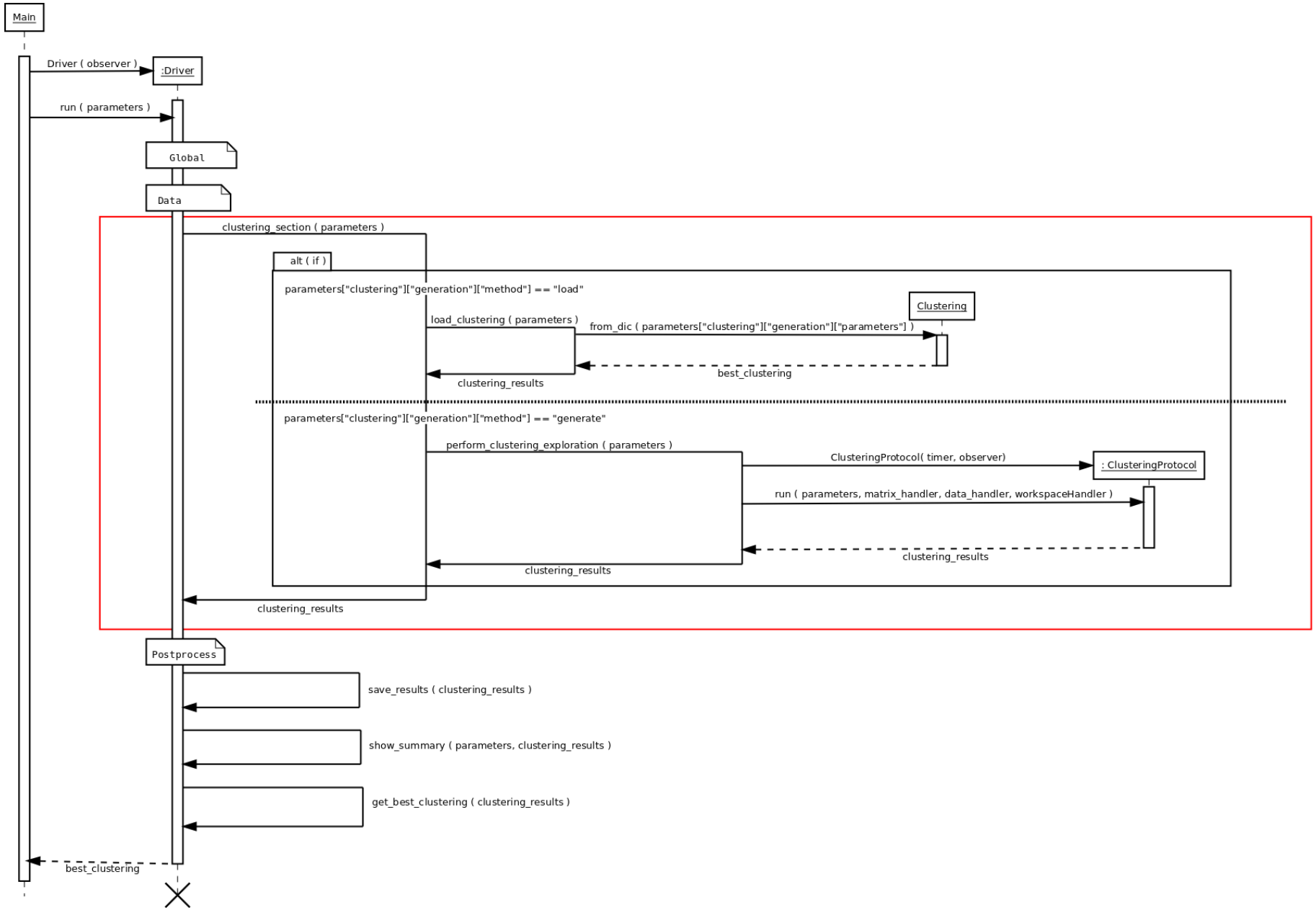

Figure 2.4: Clustering Section Execution Flow

The clustering section parameters that can be specified on the json file define if the clustering should be generated or loaded, which algorithms and parameters to use and, finally, the evaluation section which is where the user should define his goal or hypothesis.

Generation

Method, selects wether we want to load or calculate the best clustering info. If it's loaded it will use the json dictionary defined on clustering::generation::clusters.

Clusters, clustering data if the method if "load". Each cluster object must define: id, prototype and elements.

Algorithms,

for each desired algorithm to use of the six available (dbscan, gromos, hierarchical, kmedoids, random and spectral) defines it's parameters. All algorithms share the 'max', defining the maximim number of parametrizations for the algorithm, and the 'parameters' properties.

Kmedoids,

Seeding type, defines if the initial seeds should be randomly placed or at equidistant points.

Tries, if the initial seeds are to be randomly placed, this defines the number of repetitions done with different seeds (default: 10).

Spectral,

Sigma, defines the sigma parameter for the spectral clustering. If not set, the default is to calculate local sigmas.

Evaluation

Minimum clusters, minimum number of clusters a clustering must contain to be evaluated.

Maximum clusters, maximum number of clusters a clustering must contain to be evaluated.

Minimum cluster size, any cluster smaller than this threshold will be considered noise (thus increasing the clustering noise).

Minimum noise, clusterings with higher noise than this threshold won't be evaluated.

Query types, list of details to be reported about the clustering found.

Evaluation criteria, list of criteria objects, each criteria containing one or more evaluation objects.

Evaluation object, defines the sigma parameter for the spectral clustering. If not set, the default is to calculate local sigmas.

Name, defining the quality function.

Action, defines wether the function should be maximized (">") or minimized ("<").

Weight, defines the relative weight of this quality function (not mandatory that they add up to 1).

Postprocess

The postprocessing section's allows users to extract useful information about the clustering. This section is the only optional one amongst the four described.

The Driver first gets the best clustering, which is the only remaining information needed to call the PostprocessingDriver class. The run method of this class loads all the available action classes and, for each one defined on the postprocessing section of the json file, runs it with the clustering information provided. The extracted information needs to be visualized with pyProCT GUI in some cases and saved into pdb files on the others (refer to Postprocessing Parameters for more info)

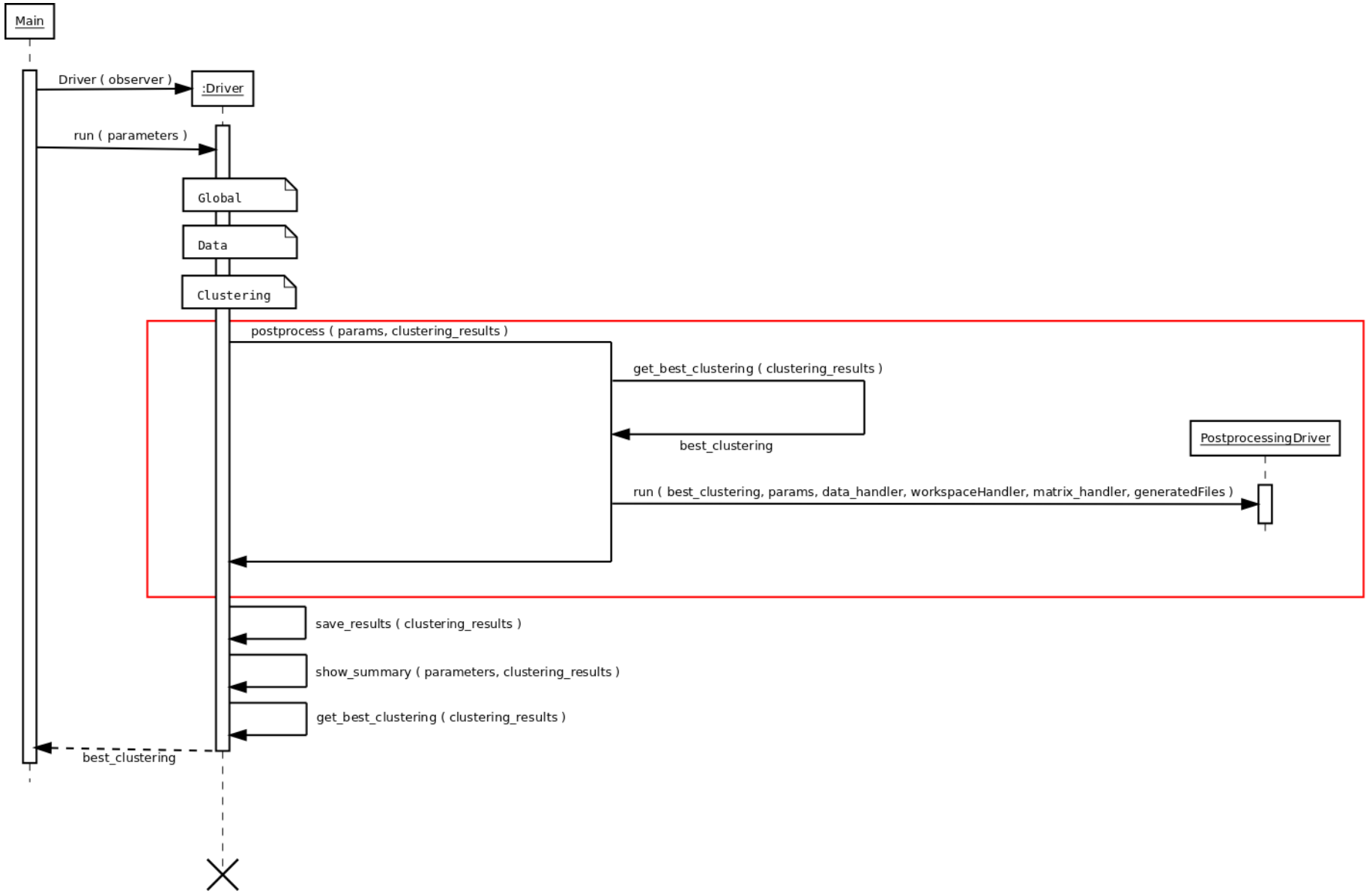

Figure 2.5: Postprocess Section Execution Flow

These are the possible postprocessing actions to be performed:

Rmsf, pyProCT will generate the global and per-cluster rmsf data to be visualized with the GUI.

Centers and trace, pyProCT will generate the data of all geometrical centers of the calculation selection of the system (to be visualized with the GUI)

Representatives, pyProCT will save the data of the medoid of each cluster on the best clustering in a pdb file.

Keep remarks, if true, stored models will be saved with the their original remarks header (default: false).

Keep frame number, if set to true, the model number of any stored conformation will match the original pdb one (default: false).

Pdb clusters, pyProCT will save each cluster information in a pdb file.

Keep remarks, if true, stored models will be saved with the their original remarks header (default: false).

Keep frame number, if set to true, the model number of any stored conformation will be the original pdb one. Default: false.

Compression, this option will produce a compressed version of the input trajectories with less redundancy thanks to the resulting clustering.

File, name of the output file withouth extension (default:compressed.pdb)

Final number of frames, number of frames the compressed file must have.

Type, sampling method for cluster elements (default: kmedoids).

Random, randomly samples elements for each cluster.

Kmedoids, uses k-medoids to get samples of the clusters.

Cluster stats, this will generate a human readable file with the distance among cluster centers and their diameters.

File, name of the generated file, without extension, to be stored inside results folder (default: per_cluster_stats.csv).

2.2 Analysis tools research

2.2.1 Motivation

After the familiarization with pyProCT code, execution flow and parameters it is time to analyze it's performance and better understand the scheduler implementation. I developed an small program designed to test the scheduler, learn to use the analysis tools, such as Paraver, find possible problems that may arise due to the parallelization, like the reproducibility on random algorithms or the results validation, and, in general, use all the required tools on a smaller scale than pyProCT.

2.2.2 Tic-tac-toe

The developed program was an command-line implementation of the popular game Tic-tac-toe. It suited quite well the project needs because it featured good parallelization options, stochastic elements, a clear turn structure (easing up the first analysis trials), variable-size parametrized tasks and automatized execution (having two AI playing).

Figure 2.6 shows a simplified version of the program's execution flow. All the logic handling which player's turn is, how the board is marked and some other parts have been omitted because they are not relevant.

For each turn, while the game is not finished, the program calls the player's play function. The algorithm implementing the AI moves is a montecarlo-like method, it randomly simulates a big number of games for each available cell and then choses the cell with the best score.

To do this it first gets all the free cells, then for each available one it calls the exploration_handler method. This method in turn calls mark_an_explore a number of times (this number is defined

by the ITERATIONS parameter which is a command-line argument) with a copy of the game board. Basically, mark_and_explore first the cell passed as parameter and then fills randomly the copied board, till the game is finished (either by a player winning or a draw), returning the cell and the id of the winning player or a 0 if there is a draw. The list of winnig id's is then returned to the montecarlo function. Afterwards we initialize the score value for each available cell with infinity. Then for each tuple containing a cell and the winner of that simulation we increment the the score of the cell if the player has won or decrease it if the player has lost (the algorithm could modify the score for draw results), the actual increment/decrement values can be tuned but for testing purposes it's not relevant.

2.2.3 pyScheduler refactor

Once the sequential program was ready, the next step was to decide how to refactor it to use pyScheduler [2], which the current scheduler used by pyProCT. The decision was to consider each exploration_handler as a task because it would be interesting to see how task's size affects the performance of the scheduler and, having the ITERATIONS parameter to change the number of iterations performed inside the exploration_handler, this was deemed the best option.

The selected scheduler has three scheduling types, one sequential and two parallel:

Serial, which executes the task sequentially.

ProcessParallelScheduler, which uses python's multiprocessing module.

MPIParallelScheduler, which uses mpi4py for the parallelization.

The usage of the scheduler is simple. For each task we have to call the *add_task* method providing the following information:

Task name: a unique task name.

[2] https://pypi.python.org/pypi/pyScheduler/0.1.0

Dependencies: a list of this task dependencies (which must be a list of other tasks names).

Description: a description of the task.

Target function: the name of the function to be executed.

Function kwargs: the list of the keyword arguments which need to be passed to the target function.

Once the task list is completed we just need to call the run() method of the scheduler. The method will return a list with the execution results of each task.

For the tic-tac-toe these are the values for the queued tasks:

Task name: ExplorationXY (being X, Y the coordinates of the cell to be explored).

Dependencies: [ ] (the empty list because there are no dependencies among different explorations).

Description: Montecarlo exploration.

Target function: self.exploration_handler (as we are inside Player class namespace we must add the self).

Function kwargs: "x": x, "y": y, "board": board (being X, Y the coordinates of the cell to be explored, and board the current state of the game).

2.2.4 Instrumenting with Extrae

Next step was to start using the analysis tools. The decision was to use the **Extrae + Paraver** combination. Extrae [3] is the package used for instrumenting the code; paraver [4] is the tool used to visualize the traces generated by Extrae. These tools have been both developed at the BSC to be used together. We chose them because of the Extrae support to python, the offered

[3] http://www.bsc.es/computer-sciences/extrae
[4] http://www.bsc.es/computer-sciences/performance-tools/paraver

assistance and proximity of the tools' experts and the fact that they are both installed and configured on MareNostrum III, which is our target execution platform.

Extrae offers two different ways to instrument the code: automatically instrument functions (providing a list of functions to the extrae XML configuration file) or including the extrae module (import pyextrae) and emit specific events inside the code with *pyextrae.eventandcounters(type, value)*.

The basic usage of the first, which does not require any changes on the code, instruments the entry and exit points of the functions; more complex behaviours are also available but for these tests the basic one is enough.

The second one just needs to add the mentioned function call wherever we are interested to emit an event.

To start, I set *play, montecarlo* and *exploration_handler* methods to be automatically instrumented and added a two events: one before the scheduler intialization and task addition and one just after the scheduler run(); on the sequential version [5] this corresponds to before and after looping through the available cells (calling *exploration_handler* on each iteration).

Support for python is only available from version 3.0 onwards and it's not fully tested so we faced some problems. For the sequential and serial versions everything went well, the traces were correct and they could be visualized with Paraver. For the parallel version we were not able to extract correct traces as extrae was not able to detect the parallelization method. When I tried the MPI version it didn't work either for two reasons. On one hand, adding the user functions' automatic instrumentation made the whole execution to end with a segmentation fault without generation the traces. On the other hand, using just the event emit method, the visualization showed just one thread.

After meeting with BSC people we managed to solve the issue. The problem was that extrae used a sequential-tracing library, so it could not detect the mpi multiple threads. To solve it

[5] From now, "sequential" will refer to the schedulerless version, "serial" to the one with the serial scheduler, "parallel" for the one using ProcessParallel and "mpi" for the mpi4py one

we substituted this sequential library with an mpi-tracing one. After some work on their part the first issue was also solved by changing some values on *Extrae_define_event_type*.

For the parallel implementation with multiprocessing this approach wasn't supported. They gave me some ideas to try but to no avail. I linked extrae with a number of different libraries, such as the pthreads one, to see it they were able to hook themselves to the python multiprocessing threads but it didn't work. I left the issue open and went forward. Fortunately, after working on the new tracing system for pyCOMPSs at the BSC, I managed to develop a workaround tracing system albeit quite more rudimentary and inconvenient to use. Extrae has a command line usage of which I used two basic commands:

extrae-cmd init node slots

extrae-cmd emit slot event_value event_type,

I created a Python class wrapper for this two commands. This way I reused it to instrument the pyProCT later. This class deals with the extrae paths, concurrency as well as providing a easier interface to call from python. To use it first it is necessary to initialize each used node with an ID and the number of processes/threads it will contain. Then for each event we want to emit we specify it's ID (which must be positive and smaller than the number of threads we set for that node) and the value and type of the event.

The first trials I dit raised a segmentation fault; knowing that emitting an event with an out-of-range thread ID raises a segmentation fault I figured out that the initialization was not correct. I tried a number of different methods. Because this extrae usage is not the recomended nor normal approach there is no documentation for it nor examples for it. After several days working on it I resolved to meet with the extrae team. Working with them we found out that the segmentation fault was caused by a bug on the release I was using. Kindly they fixed it and made a custom package for me to use. With it I finally was able to achieve a basic instrumentation for the parallel (with python-multiprocessing) version. This is limited to emit events and can not produce the advanced visualizations and results achieved on the serial and MPI version.

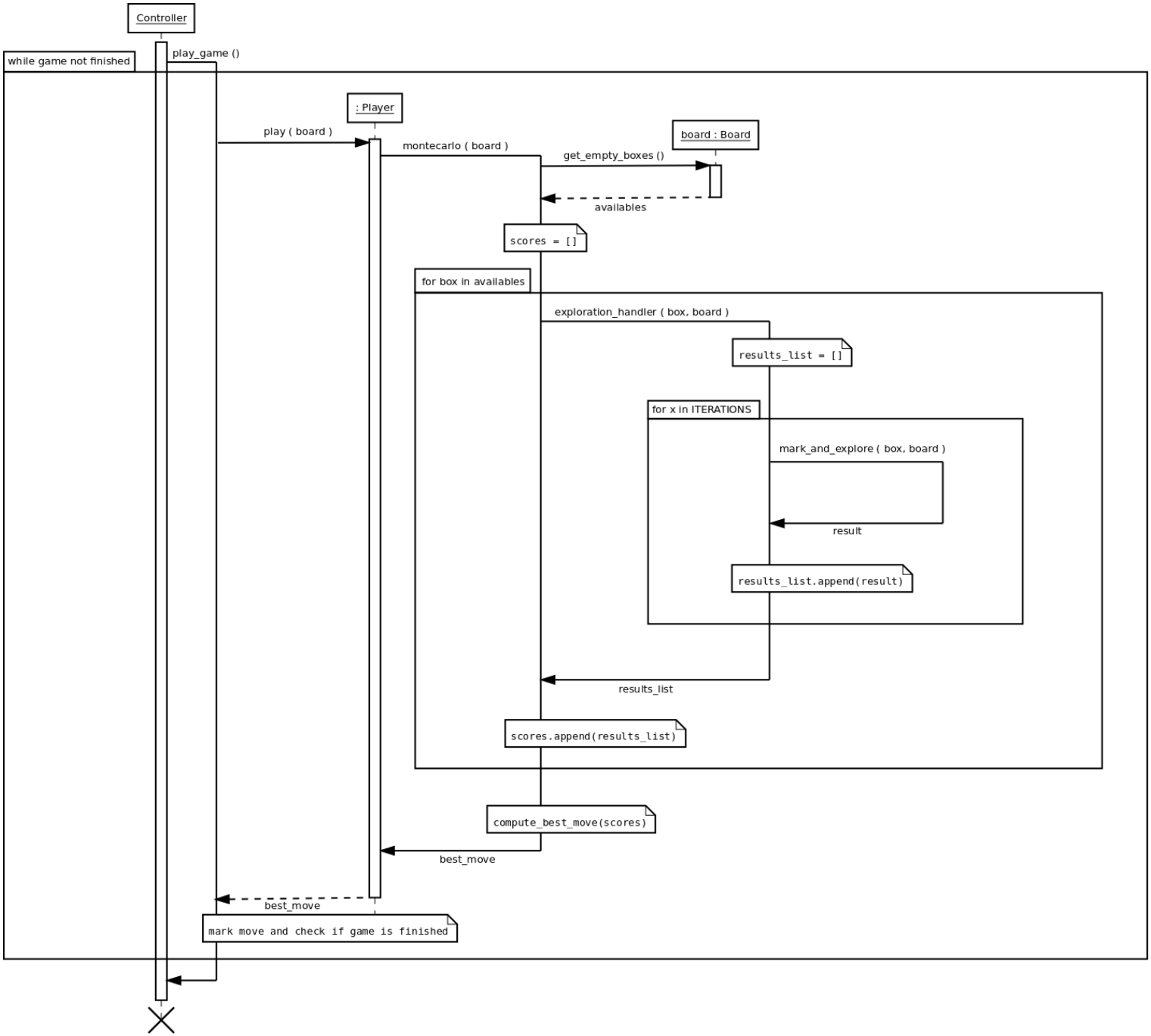

Figure 2.6: Tic-tac-toe Execution Flow

2.2.5 Visualizing with Paraver

Once generated the trace file the next step is to analyze them with the visualization tool Paraver.

First thing was to create a configuration file that would show the instrumented user functions (montecarlo, play and exploration_handler on this program). To do so I configured the event filter to show only events of the type 60000100 (which is the type assigned to instrumentated user functions), and the semantic options to show the last event value (that is the values identifying the functions) in a stacked composition. Thanks to the ability of copy/paste time info from a graphic we can quickly compare different traces.

Figure 2.7 shows the visualization of three executions of the tic-tac-toe, all of them with 500 iterations. The first is an schedulerless version, the second with the serial scheduler and the last one with an MPI scheduler. At the time of writing the parallel/multithreading scheduler can't be instrumented, as this is a demo section of the Paraver capabilities we have considered that leaving the parallel version out will not harm the purpose of this section. Dark blue corresponds with exploration_handler function, white with montecarlo, red with play and light blue the time outside these three functions; the MPI version has more labels but for the current section the details aren't important. We can see on the figure that the serial scheduler has an important overhead, making it slower than the schedulerless version, but the MPI scheduler is quite faster.

Paraver has a wide range of configurations to visualize MPI information (see Figure 2.8), useful execution time or IPC. We can inspect data in timeline or tabular form but also with the aid of other tools such as the Clustering. With this tool we can cluster the results to obtain, for example, a graphic relating the executed instructions and the IPC. Thanks to this we can bypass analysis problems related with the time where, sometimes, an increase or unbalance in the workload depends on the IPC rather than the number of instructions. On Figure 2.9 we can see that the most computation-intensive areas associated with the exploration_handler function (in blue on Figure 2.7) have a good efficiency.

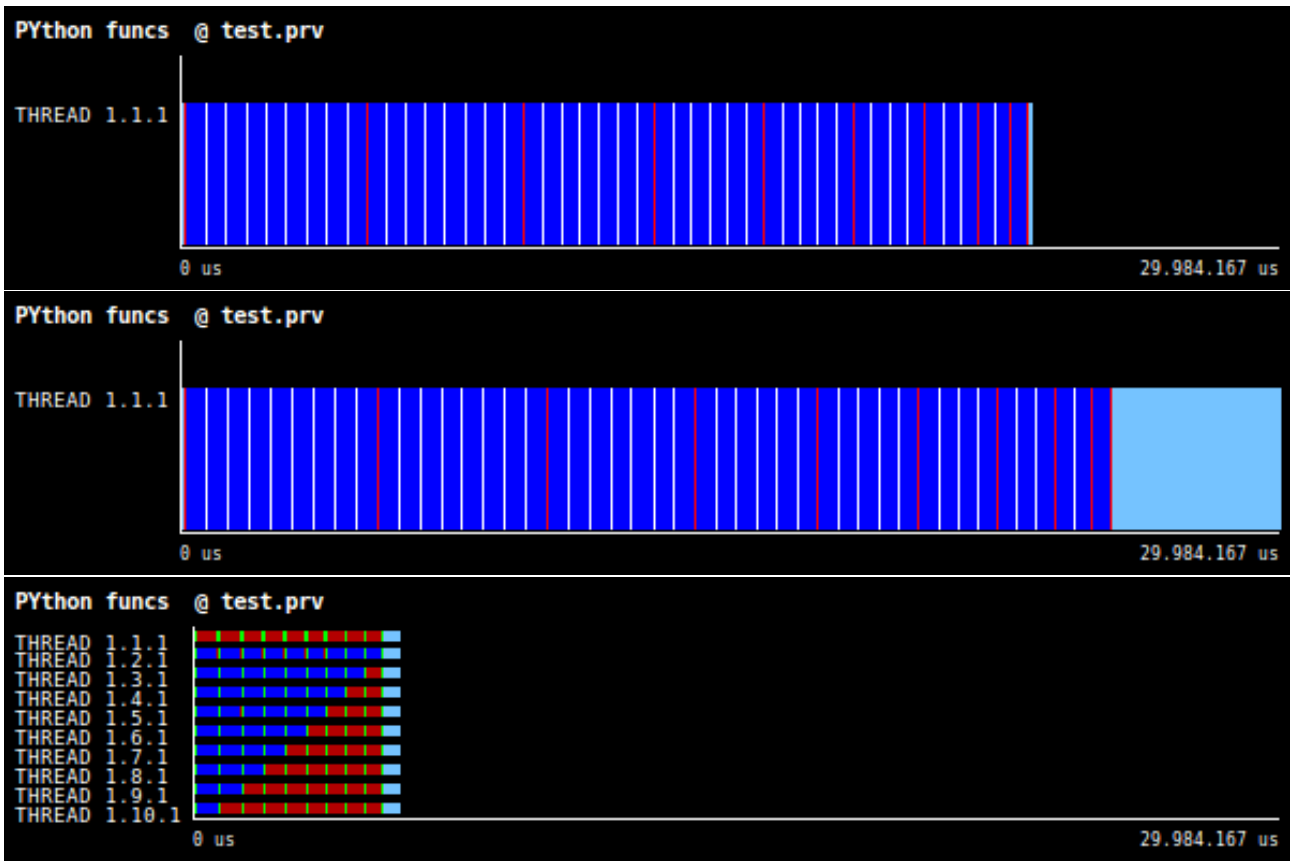

Figure 2.7: Tic-tac-toe 500 iterations executions

2.3 Validation method: pyProCT-regression

pyProCT-Regression is the software designed to validate the pyCOMPSs refactor code. It implements the so-called black-box validation method. The validator will take a list of tests to perform. First we need to generate the expected results with the original version or pyProCT, then we'll run the same tests with the new version and make sure that the output matches the expected results.

pyProCT results depend on the parameters defined on the control script because we can select which results to save, which format, wether we want to save the computed matrix (and it's image) and so on (see Section 2.1.2 for more info). Because of that the validator needs to be flexible, allowing to define more or less files to check. On the other hand, we have different two different test scenarios: one, validating the results of the original version against the refactored one, two, validate the new scheduler against known results of the other schedulers.

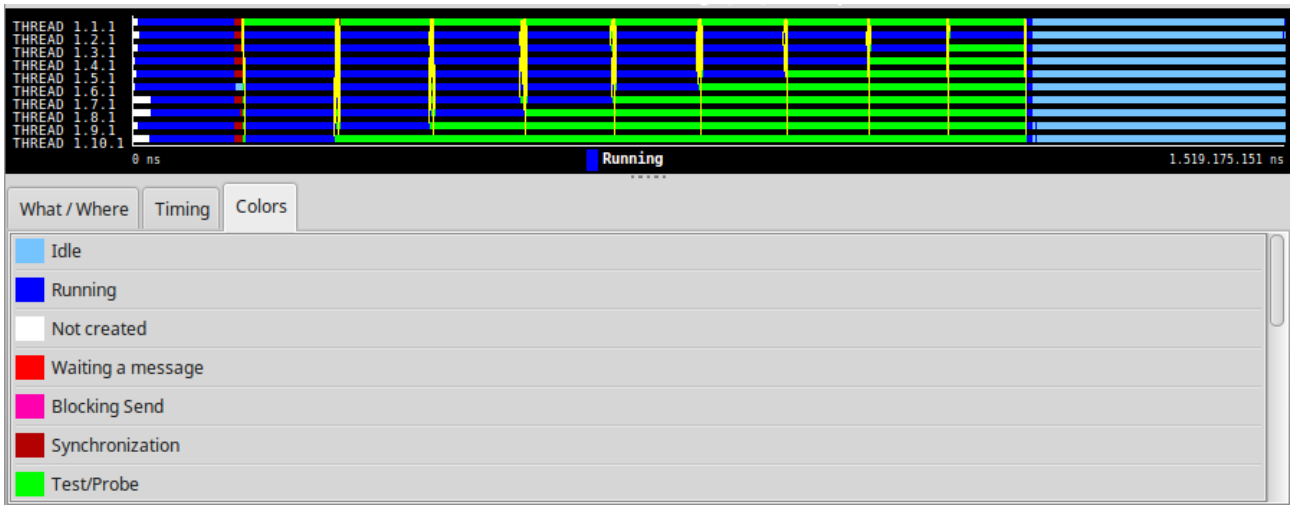

Figure 2.8: Tic-tac-toe MPI information for a 500 iteration execution

To achieve this behaviour Regression takes a test list as input with all the information it needs to check for each test scenario. Each test has a name and description and the following attributes.

Name, a unique test name.

Description, an small description of the test.

Script, defines the input script for the pyProCT execution.

Expected results dir, is the folder containing the expected output and the files specified on files_to_check

Files to check: a list of the additional files that regression will check, together with the default ones: test.out and test.err.

If we run the tester with the "GENERATE" option it will, for each test, run the installed pyProCT with the defined control script and save the normal standard output and error as well as the "files to check" on the "expected results dir".

On the other hand, running the tester with "TEST" will also run the control script with the installed pyProCT but after that it will check that the generated output matches the content of the "expected results dir".

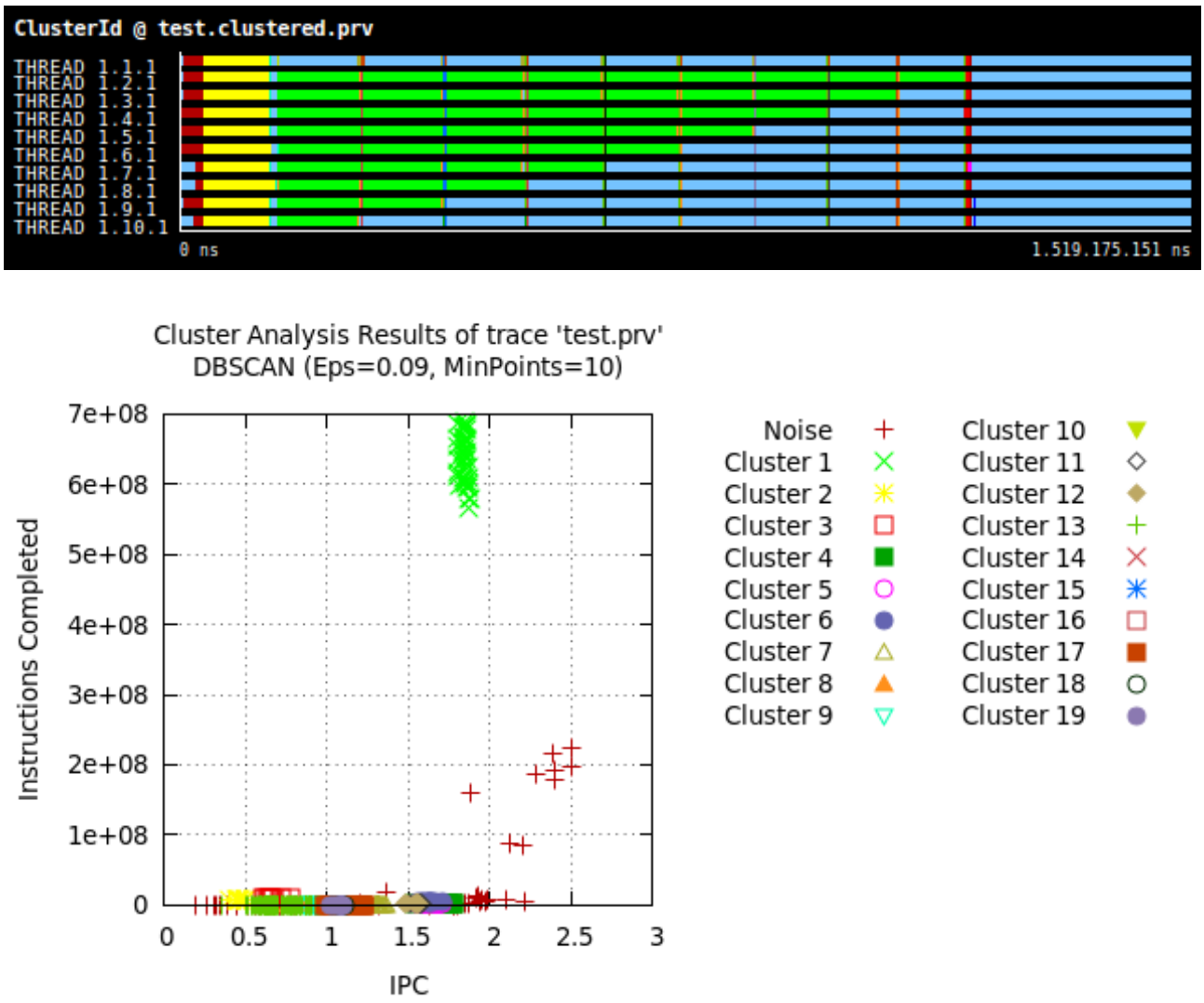

Figure 2.9: Tic-tac-toe timeline and clustering for 500 iteration MPI execution

The last three attributes allow us to use a single expected results dir with different scripts and schedulers, or use the new version of pyProCT with the same tests but on testing mode to validate the refactor. When validating that the original schedulers work as expected I also add other files to check such as the parameters.json and the clustering folders (containing information about the generated clustering).

2.3.1 Basic tests and issues

The basic tests validate each of the four sections of pyProCT (global, data, clusering, postprocess) incrementally. However once I started generating the MPI results I faced the first issue:

MPI (and later also pyCOMPSs) scheduler need to be called with mpirun and runcompss.

To solve it I could make all the schedulers work with the same call or adapt the tester to each scheduler (reading that information from the control script). I decided on the first because the scheduler is set on the control data (so the main file can act as a switch performing the runcompss of mpirun if needed) and pyProCT will be easier to use. Now the main file calls the bash script with the runcompss [params] and mpirun [params]. This way all the versions work same way and the tests on Regression just need to change the control script.

Once done this modifications it was easier to write the tests. Before starting the refactor I generated all the expected results. I tested this results with the same code that generated them to make sure the tester worked but some of them failed because of the second, and expected, issue: the random initializations of some algorithms. To solve this I did the same that on tic-tac-toe, albeit a bit more complex: "set the seeds" for the algorithm's initializations removing the stochastic and non-deterministic parts.

Afterwards, for each scheduler I ran the basic tests, first with the original pyProCT, then with the refactor. However the pyCOMPSs mode was not available before the refactor so I validated it's output against the expected results of the MPI (could be any of the others). pyCOMPSs embeds the application output on its own so, in this case, the tester checks if the expected output is contained within the pyCOMPSs one.

Finally for each major modification of pyProCT I ran this suite of tests to validate the work done.

2.4 Refactor

This section will walk you through all the refactor process. It will provide a full description of the issues found, wether they were solved or not, the design decisions made and the reasons behind them, and all the information relevant for debugging, testing and further developing both pyCOMPSs and pyProCT.

2.4.1 Set up

The installation of pyProCT as described on the pyProCT repo is trivial on a local machine. On MareNostrum III pyProCT is already installed. In order to use my version under development (instead of the package installed both on MN3 or a local machine) the user just needs to point the python path to it. This is useful to switch between different working versions (for example to use pyProCT-regression validation or to meet the different instrumentation requirements of each scheduler). Later some issues will also force me to use this same method to customize some of the dependencies of pyProCT such as the pyScheduler or pyRMSD.

Once I installed and ran a few pyProCT examples on my local machine I proceeded to MareNostrum III to do the same. Choosing a good structure to set up the environment is a must for executions on MN3.

I faced and spent a lot of time on configuration problems. This kind of issues kept popping up during the whole project. Despite that I prefered to group them here and give a brief description of the issues and how they were solved.

This first issue arose when trying to compile and link the development version of pyProCT. It is related with MN3's modules environment. By default, on login, MN3 has 2.6.9 python, however this version is not available to be loaded through the modules; it's only available when no other python has been loaded by the .bashrc file nor manually with *module load PYTHON*.

pyProCT depends on python 2.7.3 which can be loaded with the modules. At first I compiled and installed it with the default release (2.6.9) with setup.py. On MN3 I had to add a custom installation path (with *–prefix=PATH* option) to setup.py because I have no permissions to write into the default installation directory. After installing it I found out that pyProCT can not be run under python 2.6.9 so I started again all the installation process with 2.7.3 once I figured what was causing the error.

The new installation lead to the following new bug:

undefined symbol: PyUnicodeUCS4_DecodeUTF8

This is caused when trying to use software build with UCS4 on a UCS2 python version. On MN3 each installation uses a different one.

- Python 2.7.3 → UCS2
- Python 2.6.9 → UCS4

Python is not a compiled language, so this compilation problem actually comes from the Cython modules integrated into pyProCT. This meant that the new installation (which used the same folder as source) was not recompiling the Cython modules even after issuing a clean command so I cloned the repo again and started from scratch. This time everything ran smoothly. As a curiosity if pyProCT is build with python2.6.9 it can be used with python2.7.3 (although rising some compatibility warnings).

2.4.2 pyCOMPSs

Prior to starting the refactor I analyzed which would be the best way to parallelize it. pyCOMPSs works by using python's decorators to define some functions as *COMPSs' tasks*. These tasks are executed on previously defined resources such as a MN3 node or a cloud. For each task the framework checks wether that function's parameters depend on some previous task; if it has no dependencies then the task is assigned to a resource which runs it.

pyProcT clustering and postprocessing sections, as previously stated, are embarrassingly parallel: all algorithm's executions depend only on the distance's matrix calculation; the postprocessing actions all depend on the best clustering (that is to say: the whole clustering section). Knowing this we decided to define as task each algorithm execution and each postprocessing action.

I wanted to maintain the possibility to use the other schedulers after the refactor so I kept the overall structure of pyProCT. However, I also wanted to exploit the possibility of reduce the code complexity while achieving the maximum performance improvement. I mention this because make the sequential version of pyProCT work with pyCOMPSs is enough to place

the decorators on the right functions. It is true that this would also raise some issues to be addressed; my point is that the lines of code required are few if the goal is just to make it work. This is not the goal though. The refactor described from here onwards tries to minimize the code size, make it clearer. It also removes functionality duplication between the framework and the software. For example pyProCt has a loop which enqueues the tasks for the scheduler. COMPSs also has an internal queuing system rendering this loop unnecessary.

Bearing this in mind, differences are basically found on the Driver, Protocol and Explorer classes, which deal respectively with all the sections pipeline execution, the clustering pipeline, and the clustering exploration *per se*. I simply created a new Driver for the COMPSs scheduling; the main checks whether pyCOMPSs is the scheduler or not and calls one driver or the other accordingly (same method being used for MPI). From the driver onwards the key classes are substituted by the COMPSs versions.

One of the advantages of pyCOMPSs is the small amount of work required to use it. On a normal sequential program we just need to use the *@task()* decorator and the *obj = compss_ wait_ on(obj)* API call to create synchronization points for future objects; from COMPSs manual:

> If the programmer defines, as a task, a function or method that returns a value, that value is not generated until the task is executed. However, in order to keep the asynchrony of the task invocation, COMPSs manages future objects: a representant object is immediately returned to the main program when a task is invoked.

Internally COMPSs has queue of tasks so the step to add the tasks to the scheduler is no longer required; instead I called directly the decorated methods (which internally COMPSs enqueues to it's pending's list). However this caused a problem related to the how the framework deals with the data.

COMPSs is a framework which allows to define a lot different resources. The communication layer needs a high level of abstraction because workers (resources able to execute tasks) use different protocols (e.g. SSH or NIO). To send the data needed by each task, that is, the method's parameters, COMPSs serializes them to Java objects (except basic types). This

means that python's pickle must be able to do the translation which is not the case for the distances matrix.

PyProCT uses pyRMSD to represent the forementioned matrix. It is basically a python wrapper for a C matrix structure. The goal of this implementation is to highly reduce the access time to the matrix elements.

> Python is slow [...], why: it boils down to Python being a dynamically typed, interpreted language, where values are stored not in dense buffers but in scattered objects. [1]

To overcome this Víctor wrote the lean and specialized pyRMSD (which stands for python Root Mean Squared Deviation). The problem is that these structure is not a native python type nor it's built with a combination of them. Because of this the framework can not serialize this matrix to send it to each worker. The first idea to solve it was to dump the internal data of the matrix into a python list (which can be serialized arbitrarily) with the already implemented method *get_data*; this list then can be used to reconstruct the structure the class constructor *CondensedMatrix()*. This approach raised two issues.

The first issue was where to perform the translations to a python list, which is linked to which is the function decorated as task. Before presenting the solution and the next issue it is worth to make a point about how the algorithms are implemented and called.

Each algorithm is implemented in a different class so we have *kMedoidsAlgorithm.py*, *spectralClusteringlAlgorithm.py* and so on. All of them however return the same kind of data: clusterings; in order to simplify all the execution pipeline and the code they all have the same structure: all the required arguments are passed to the class constructor and then they all have the *perform_clustering* method which returns the clusterings found for that parametrization.

With this structure then it would be necessary to decorate the *perform_clustering* method of each algorithm but only when pyCOMPSs scheduling is used. In order to avoid having code duplication (one with the decorator and one without for each algorithm) I implemented

a wrapper class, called CompssTask, for the algorithm's execution with a single pyCOMPSS-decorated method.

The class is constructed with all the required information for the task execution. During the initialization I also do the forementioned translation of the matrix to a python list. After the constructor the run method, which is the actual pyCOMPSs task executed on a worker, recreates the Condensed Matrix from the python list and executes the algorithm's clustering method.

After the execution of the algorithm it was necessary to again deassign the computed matrix because it is part of the class and, even if it's not a result, when the task is finished pyCOMPSs again tries to serialize the object and fails.

2.4.3 Other issues

When trying to generate traces with extrae (for MPI and sequential version) I got an Prody error. When trying to resize any kind of structure Prody detects that there is more than one reference to that structure (introduced by the instrumentation) and fails to do the resize. To avoid this I had to manually modify the Prody package and, for each resize, add the parameter *refcheck=False*. This error is raised in order to avoid integrity problems when an object has more than one reference; however we know that the instrumentation will not modify nor actively use those structures so we can safely disable the reference's check.

Another issue was raised by some datasets. Depending on the computer and data when I try to recreate the condensed matrix on the pyCOMPSs task I get an incompatible format error. This happens when numpy stores the matrix data (in list format) as floats with 32 bits. The matrix constructor however requires that data to be in floats with 64 bits. To overcome this I found that numpy arrays have a method, *data_ view('float64')*, to select which type of elements should be returned and thus allowing me to always format them as 64-bit floats and solve the issue.

3. Results

This section describes the results of the refactor described in the previous section.

3.1 Performance

3.2 Usability

COMPSs programming model aims to ease the development and execution of applications. In this section I will evaluate these goals.

First I will talk about the code refactor. As stated in 2.4.2 pyCOMPSs I tried to reduce the code to keep it simple. COMPSs is designed for people without parallelization knowledge, and thus, for sequential programs. I wanted to exploit the simplicity of COMPSs. In order to do it I emulated the process of writing the code directly for pyCOMPSs from a sequential program. This is the scenario I will first analyze.

When talking about the complexity of a program one the first questions is about the code's size. Figuer ADDDDDDDD REFERENCE shows the comparison, in number of lines, between the duplicated classes (the ones used on the original schedulers and the modified versions used when pyCOMPSs scheduler is selected).

Class	Original	pyCOMPSs
Developer	10,00 €	664h
6.640,00 €		

Table 3.1: Human Resources Budget

Python is a language designed to be easy to read with strong coding conventions so I did not try to minimize the code. I coded in a normal fashion trying to make things clear. Bearing this in mind let's see how well fared the refactored code against the original one:

3.3 Miscellaneous

4. Conclusions

5. Glossary

This section describes all the technical terms used and provides a list to the documentations mentioned throughout this document.

Cluster analysis(CA), it the task of grouping a given set of objects in way that items on the same group or cluster are more similar (in some sense) to each other than to those in other groups.

Pip, is a package management system used to install and manage software packages written in Python.

Cython, programming language is a superset of Python with a foreign function interface for invoking C/C++ routines and the ability to declare the static type of subroutine parameters and results, local variables, and class attributes.

COMPSs, is a programming model which aims to ease the development of applications for distributed infrastructures. It features a runtime system that exploits the inherent parallelism of applications at execution time.

Decorator (python), is a function that takes another function and extends the behavior of the latter function without explicitly modifying it.

Pickle, is the python standard mechanism for object serialization; pickling is the common term among Python programmers for serialization (unpickling for deserializing).

Pragma, directives offer a way for each compiler to offer machine- and operating system-specific features while retaining overall compatibility with the C and C++ languages.

pyCOMPSs, python application programming interface (API) for COMPSs.

MareNostrum III (MN3), is a supercomputer based on Intel SandyBridge processors, iDataPlex Compute Racks, a Linux Operating System and an Infiniband interconnection located at Barcelona.

MN3 Modules Environment, is a package (http://modules.sourceforge.net/) which provides a dynamic modification of a user?s environment via modulefiles. Each modulefile contains the information needed to configure the shell for an application or a compilation. Modules can be loaded and unloaded dynamically, in a clean fashion

MPI, is a standardized and portable message-passing system designed to function on a wide variety of parallel computers.

Non-blocking I/O (NIO or "New I/O"), is a collection of Java programming language APIs that offer features for intensive I/O operations.

Open Multi-Processing (OpenMP), is an API that supports multi-platform shared memory multiprocessing programming. It consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior

ProDy, is a free and open-source Python package for protein structural dynamics analysis. It is designed as a flexible and responsive API suitable for interactive usage and application development.

Root-mean-square deviation (RMSD), is the measure of the average distance between the atoms (usually the backbone atoms) of superimposed proteins.

setup.py, is a python file, which usually tells you that the module/package you are about to install have been packaged and distributed with Distutils, which is the standard for distributing Python Modules. Allows to easily compile and install with *python setup.py build && python setup.py install.*

Secure Shell (SSH), is a cryptographic (encrypted) network protocol to allow remote login and other network services to operate securely over an insecure network.

Unicode, is a computing industry standard for the consistent encoding, representation, and handling of text expressed in most of the world's writing systems.]

UCS-2 & UCS-4, are Unicode encodings which encode each code point to exactly one unit of, respectively, 16 and 32 bits.

Wrapper, function (or class) is a subroutine in a software library or a computer program whose main purpose is to call a second subroutine or a system call with little or no additional computation.

Framework, is often a layered structure indicating what kind of programs can or should be built and how they would interrelate. Some include actual programs, specify API's, or offer programming tools for using the them.

.bashrc, is a shell script that Bash runs whenever it is started interactively (when login into MN3 for example).

6. Documentation

- **pyProCT**

  Github Readme,

  https://github.com/victor-gil-sepulveda/pyProCT

  Dropbox Supporting Information,

  https://dl.dropboxusercontent.com/u/58918851/pyProCT-SupportingInformation.pdf

- **pyScheduler**

  Github Readme,

  https://github.com/victor-gil-sepulveda/pyScheduler

  Python Package Index (pypi),

  https://pypi.python.org/pypi/pyScheduler

- **MareNotrum III**

  User's guide,

  http://www.bsc.es/support/MareNostrum3-ug.pdf

- **COMPSs**

  User Guide,

  http://compss.bsc.es/releases/compss/latest/docs/compss-manual.pdf?tracked=true

  Tutorials,

  http://compss.bsc.es/releases/tutorials/?tracked=true

  IDE User Guide,

  http://compss.bsc.es/releases/ide/doc/1.2/COMPSs_IDE_user_guide_v1.2.pdf?tracked=true

Installation,

http://compss.bsc.es/releases/compss/latest/docs/installation-guide.html?tracked=true

Release Notes,

http://compss.bsc.es/releases/compss/latest/docs/RELEASE_NOTES?tracked=true

- **Performance Tools**

Extrae User Guide,

http://www.bsc.es/sites/default/files/public/computer_science/performance_tools/extrae-3.1.0-user-guide.pdf

ClusteringSuite intro,

http://www.bsc.es/ssl/apps/performanceTools/files/docs/T2_Clustering.pdf

ClusteringSuite manual,

http://www.bsc.es/sites/default/files/public/computer_science/performance_tools/clusteringsuit

Paraver introduction,

http://www.bsc.es/sites/default/files/public/computer_science/performance_tools/w1_introtool

Paraver internals and details,

http://www.bsc.es/ssl/apps/performanceTools/files/docs/W2_Paraver_details.pdf

Instrumentation,

http://www.bsc.es/ssl/apps/performanceTools/files/docs/2A_Instrumentation.pdf

Tools scalability,

http://www.bsc.es/ssl/apps/performanceTools/files/docs/T1_Scalability.pdf

m

7. Temporal Planning

7.1 Overview

This document describes the temporal and resource planning for the project pyProCT optimization. The project was started on 15th February, 2015 and its deadline is June 2015.

7.2 Task List

This project is going to be developed using an SCRUM methodology so the mentioned times are related to the duration of the work cycles.

As mentioned on the Project Scope document, each cycle is formed by a prior analysis, the optimization implementation and, finally, the results validation. Each cycle is independent from the other ones, even if they do affect each other, so there are no precedence dependencies. However, the code refactoring using COMPSs is going to be the first and more important one because not having a good parallelization could shadow the improvements introduced by task-level optimizations.

With this in mind we set up the following task list:

1. Project Management

    1.1. Project's Scope - 9h

    1.2. Temporal Planning - 6h

    1.3. Budget and sustainability - 3h

1.4. Preliminar Presentation - 6h

1.5. Context and Bibliography - 15h

1.6. Specialization specification - 10h

1.7. Final document and presentation - 20h

2. Code familiarization - 40h

3. Analysis tools' research - 40h

4. Common set up for all SCRUM cycles - 40h

5. SCRUM iterations - 400h control cycles

5.1. Initial analysis and optimization decision

5.2. Optimization development

5.3. Implementation measurements

6. Global performance analysis - 40h

7. Project Writing and Defence Preparation - 80h

7.3 Tasks Description

7.3.1 Project Management

This task is the responsible for the whole project planning and specification and covers all the deliverables of the GEP course.

Resources list

TexStudio,

desktop application used for report writing

GanttProject,

desktop application for Gantt chart creation

UPC Atenea,

online platform for deliverables submission

7.3.2 Code familiarization

The project aims to optimize an already existing project. This means that the first thing to do before starting to work is to explore, execute and, in general, familiarize myself with the original code.

Resources list

Git and Github,

to use the code we need a Github account to fork the original project repository. Once forked we need a git-able OS, in this case Linux Mint 17, to clone it.

MareNostrum III account,

for executing the code on the supercomputer and correctly assess it's performance as well as execution limitations for instrumentation purposes.

SSH-able OS

to establish secure shell connections to the MareNostrum III computer for the program execution.

Paraver and other analysis tools,

to ease the understanding process with execution diagrams. Also, on this first stage, I will start looking for the best available tools for the posterior analysis stage.

7.3.3 Analysis tools' research

Once the whole program execution and mechanics, the next step is to decide which analysis tools are going to be used. Paraver is going to be amongst them. This task needs to be done after getting familiar with the code because otherwise we could end up trying to use tools which are not compatible with the python version and packages used and the remote execution pipeline.

It is important to note that this is mainly a research stage. This means that no consistent code modifications are going to be performed, just the necessary ones to ensure that the tools work well with the code.

As a big part of the project is going to be analysis, we set up first this research stage and an implementation/instrumentation one in order to correctly address the importance of this matter.

Resources list

See Code Familiarization resources list 7.3.2.

7.3.4 Common set up for all SCRUM cycles

Once I am familiar with the code and the analysis tools have been chosen, the next step will be to instrument the code for it's preliminary analysis. The instrumentation should be deep enough to test all the possible implementations to be done, regardless they are performed on the matrix distances calculation, task level or scheduler level.

On this stage we will also do a preliminar work aimed to automate and speed up as much as possible the code analysis and execution. On one hand, this will speed up the SCRUM iterations by automating the analysis and execution with tools like bash scripts, allowing us to focus on the actual development and analysis and avoid wasting time on repetitive task as

graphs' generation or the remote execution. On the other hand, it will also help to avoid human errors on the execution parameters, analysis settings and environment configuration.

Resources list

The resources required for this stage are linked to the analysis tools decisions so they can not be listed until the first stage is finished.

However common resources for automation are required (i.e. bash scripts). The Linux Mint environment used provides this basic functionalities.

7.3.5 SCRUM iterations

As explained on the Scope of the Project document, the work flow will follow the analysis-implementation-analysis structure. Each cycle is going to have, at least, one control meeting for each three weeks of work. This way we will keep track of the optimization development and solve possible problems.

The first and major implementation is going to be the COMPSs refactoring. Being the most important optimization, a maximum period of two months is going to be assigned to this iteration. Because the other optimizations are going to be decided at the start of each cycle no more constraints are going to be applied to them, instead they will be decided at the start of each cycle.

The work to be done on each phase of the iteration has already been specified on the Project's Scope document and requires no further description.

Resources list

The common requirements for each cycle are all the ones mentioned on the previous sections. Some optimizations may require new tools and resources so at the start of each iteration, on the preliminary analysis, a resource analysis and specification is going to be performed.

7.3.6 Global performance analysis

After finishing all the implementations decided, a global analysis is going to be performed. The aim of it will be to show not the changes introduced by each optimization but how these all behave and interact between them. Also the global speed up obtained, the final execution pipeline and conclusion fall into this section.

Resources list

This section requires no more resources than the needed to correctly reach this stage of the project.

7.4 Gantt and PERT charts

This is the resulting Gantt and PERT charts for the described tasks.

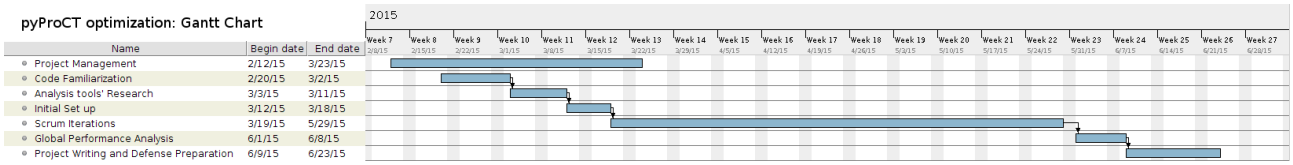

Figure 7.1: Gantt Chart

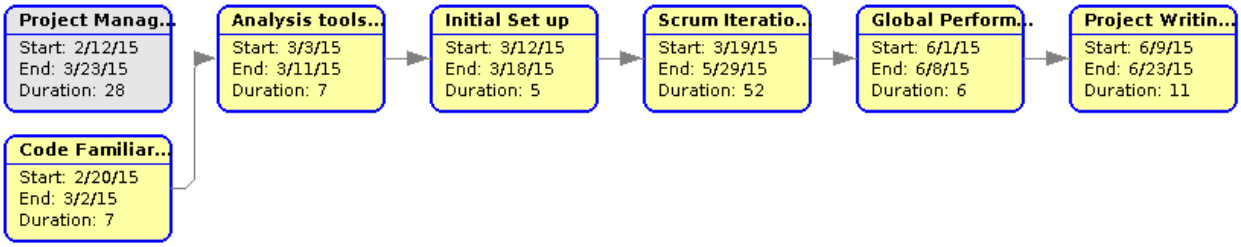

Figure 7.2: PERT Chart

8. Budget

This section describes the required budget for the pyProCT optimization project. It contains a detailed description of the material and human costs. The description is divided into: human, hardware and software resources; instead of specifying the costs per tasks, we have decided to use this structure because there are not remarkable differences on the resources used for each task so grouping them like this the document will be clearer, will avoid too many subsections and, on the Temporal Planning, the resources needed for each task have already been specified.

8.1 Human Resources

This project will be designed, developed and evaluated by one person so no roles are going to be present on the analysis. The estimation of the human cost is tied to the work time represented, in this case, by the Gantt Chart and the task description provided on the 7.4 Gantt and PERT charts section of the temporal planning.

The project will be completed single-handedly developer with an eight-long workday from Monday to Friday, without having holidays.

First is important to note that the Project Management task overlaps with other tasks. However the duration of this section is tied to the programmed schedule of the GEP course, not to the amount of work required to finish it. Thanks to that, we will consider that from 12th February to 12th March the workday is going to be equally distributed amongst the overlapping tasks, which are less work-intensive because they are merely familiarization and research tasks.

Taken that into account, the defined work period has 83 workdays from 12th February to 4th June (12d + 22d + 22d + 22d + 5d monthly breakdown) amounting to a total of 664 hours.

So the human cost will be:

Role	Price per hour	Time	Cost
Developer	10,00 €	664h	6.640,00 €

Table 8.1: Human Resources Budget

8.2 Hardware Resources

The hardware resources for this software project are going to be the development device, a laptop, and the testing one, the Mare Nostrum III. It's assumed that the computer used for development has an internet connection and electrical connection. These costs are covered on the total budget together with unexpected costs. However to reduce the budget one possibility would be to consider using the university facilities. The university provides to it's students and developers a free network and plugs which is more than enough in this case.

Product	Price	Useful life	Amortisation
Mare Nostrum III	22.700.000,00 €	3 years	0[1] €
Laptop	1.200,00 €	3 years	150,09[2] €
Total	22.701.200,00 €		150,09 €

Table 8.2: Hardware Resources Budget

8.3 Software Resources

pyProCT is an open source software hosted on a public github repository which can be used without restriction subject to the condition of citing the following article:

[1] MareNostrum III is a public infrastructure so users need not to pay to use it

[2] Given by: Cost / Useful life * Time used on project (664h)

Copyright (C) 2012 Víctor Alejandro Gil Sepúlveda pyProCT: Automated Cluster Analysis for Structural Bioinformatics J. Chem. Theory Comput., 2014, 10 (8), pp 3236–3243 DOI: 10.1021/ct500306s

As our aim is to improve this software we want to keep it as it is. This means, on one hand, that all the features and optimizations added to it will also be free and public, using no third-party paying software. On the other hand, being it a public software we have decided that the development will allow reproducible research, meaning that all the tools used for analysis are also going to be free and available to anyone trying to reproduce the analysis and optimizations of this project.

Product	Price	Useful life	Amortisation
Linux Mint 17.1	0	-	0
Extrae	0	-	0
Paraver	0	-	0
Git	0	-	0
Github account[3]	0	-	0
Texstudio	0	-	0
GanttProject	0	-	0
Dia2code (UML drawing)	0	-	0
Atenea UPC	0	-	0
Other tools	0	-	0
Total	0 €		0 €

Table 8.3: Software Resources Budget

[3]The repository is public so no premium account is required

8.4 Total Budget

Adding up all the cost described on the previous section we get total cost of the project, to which we need to add the VAT, which is 21 % in Spain. We do not expect big problems or incidents because, as we stated, we aim to use only free software so any modification or change on the task's planning will mainly just add office rental costs (taking into account that the office rental also includes the electricity and internet costs).

To control unexpected events we will add to the Total Cost an amount of money to confront them. These would cover various problems such as: an electricity or internet cost rise, more required office time (rising the rental costs and network/electricity) or, in case of not having enough time, the hiring of supporting help (other developers).

Product	Price	Useful life	Amortisation
MareNostrum III	22.700.000,00 €	3 years	0[4] €
Laptop	1.200.00 €	3 years	150,09[5] €
Office rental	5.000,00 €	-	5.000,00 €
Unexpected costs	3.000,00 €	-	3.000,00[6] €
Subtotal	30.701.200,00 €	-	8.150,09 €
VAT (21 %)	6.447.252,00 €	-	1.711,519 €
Total	37.148.452,00 €	-	9.861,609 €

Table 8.4: Total Budget

[4]MareNostrum III is a public infrastructure so users need not to pay to use it

[5]Given by: Cost / Useful life * Time used on project (664h)

[6]Given by: Cost / Useful life * Time used on project (664h)

9. Sustainability

9.1 Economic

An economic assessment has already been described on the Budget Section. The resources used for the development have been kept to a minimum; trying to use all the free software and resources provided by the university, in which a project like this could be developed; being developed by a single person, implying just one salary. The time used, however, could be reduced by having a developer and an analyst. This way on the SCRUB iterations while the developer is working a new feature the analyst could be working on the results of the previous one and so on. A part from that it's difficult to reduce more the amount of work because we have already considered a full-length workday without holidays.

On the major optimization, the COMPSs refactor, we are utilising an already developed framework which eases a lot the amount of work required for it. Thanks to the collaboration with the COMPSs developing team we achieve, as said, a faster implementation, good support, because the framework is currently used, and also we help a good framework as COMPSs to gain more notoriousness and relevance on research projects.

This project will have an 8 in the economical viability area because the cost can hardly be reduced. However, performing a more in-depth ot the risks of the project could help to reduce the budget for unexpected problems.

9.2 Social

On the social dimension we find that the optimization of this software will allow more research teams or enterprises to use it. This is important because on large datasets the amount of time and processing power can be overwhelming for small teams. Even if an optimization as this can not directly change a whole country, it could help universities and developers to waste less time of Supercomputing centers which are quite expensive and so improve their performance and resource disposal and use. As said, these project will not produce better results for the end user but will help the HPC providers.

The optimization covers a necessity as this project is done on BSC demand. So they will benefit from it, harming no one else on the process.

On the social area it will also receive an 8 because it will help to improve and further develop the COMPSs framework on the industry, providing more data, cases of use and information to the BSC/COMPSs development team.

9.3 Environmental

The environmental-related resources of this project are, primarily, two: the Mare Nostrum III supercomputer and the development laptop.

Both resources use electric energy. Theoretically this project will diminish the energy used by the supercomputer, which is quite high, but, in fact, the supercomputer is always running so the impact will be almost negligible, from a power-consumption point of view. However, on smaller scale devices this could in fact reduce a little the amount of energy spent, not worsening it in any case.

The COMPSs framework is used to help the development so we are reusing previously done work, giving the original project more scope and giving it more usage, helping to make the most out of the framework development.

This a software project so no other resources than the electricity used to run it is required. No direct waste is going to be produced by it's use. As it is an open source project aimed to be used or improved the whole project can be reused, both for new projects aiming to reduce even further the power and time consumption and for teams requiring a generic clustering analysis method.

On the resources analysis the project will be awarded with a 9 because the only thing that it's not environmental friendly is the electricity consumed by the Mare Nostrum III and the laptop to run the program and, compared to the average consumption per person nowadays, it's not a big deal.

9.4 Context

Nowadays the amount of digital information available is exponentially increasing. Just on 2015 we generated almost 8.000 exabytes of information. Facebook generates 105 terabytes of data each half hour, more than 48 hours of video per minute are uploaded to youtube and google has at least 1 million queries/minute. But why do we observe this massive increase? To start the cost of creating, managing and storing information has dramatically dropped: EMC Corporation estimates that on 2011 this cost has been cut to a 1/6 of what it was on 2005. But more importantly people is more connected than it has ever been; mobiles, websites and social channels are just some examples of a whole new world of data-generating people interactions.

In this scenario is where we found the hot topic of today: Big Data. So what is it?, usually the term is used referring to data sets too big or complex to be processed with traditional data applications or on-hand management tools. According to the IT giant Gartner, Inc Big Data can be characterized by the "3 Vs", velocity, volume and variety [**?**]:

> "Big data" is high-volume, -velocity and -variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making.

However all this raw information needs to be processed and categorized in an effective way before being used. Cluster analysis methods are one of the most used tools to address this issue.

The term **cluster analysis** (first used by Trion, 1939) refers to the task of sorting similar objects of a data set into groups (called clusters) in a way that the degree of similarity between each pair is maximal if they belong to the same cluster and minimal otherwise. Data sets can be imagined as points in a multidimensional space, where each feature of an object would represent a dimension. The CA methods need to identify, as efficiently as possible, the denser areas and group the into clusters.

Thanks to the clustering we can reduce the size of large data sets by extracting the most relevant information, usually the common features of a group or a subset of representatives. Cluster analysis (CA from now onwards) techniques thrive in the Big Data world because it's not feasible to manually label objects, there is no prior knowledge of the number and nature of the clusters and, also, their identifying traits may change over time.

It is important to note that cluster analysis it's not an specific algorithm but rather the general task to perform. Due to the fact that the similarity criteria it's subjective and can change a lot between data sets, there isn't an optimal clustering algorithm. This is the reason why there are so many clustering algorithms, each with it's advantages and inconveniences. Each algorithm uses it's own kind of cluster model that defines how the algorithm groups the items and defines the clusters. Some of the most relevant examples are:

Hierarchical Clustering Analysis (HCA)

These methods seek, as their name indicates, to build a hierarchy of clusters. These can be done by starting with all elements in one clusters and the divide them in a "top down" way, this method is called **Divisive**. Opposed to this one we find the **Agglomerative** method, where each data point starts in a different cluster merging them as one moves up the hierarchy.

Centroid Clustering,

On these algorithms the similarity between different clusters is defined as the similarity between their centroids. **K-means** clustering is one of such methods, on it, each observation belongs to the nearest centroid which, in turn, serves as the representative or prototype of the cluster.

Distribution-based Clustering

Clusters are modelled by statistical distributions. On this category falls the well-known **expectation-maximization (EM) algorithm** which uses multivariate normal distributions.

Density Clustering,

These methods follow the intuitive notion, described earlier, of considering the observations as clouds of points in a multidimensional space and so, they identify clusters as connected dense regions in the data space. **DBSCAN** is one of such algorithms and it's both one of the most common as and most cited in scientific literature.

It is also possible to classify clustering methods by some other properties such as:

Hard Clustering,

where each element belongs to a cluster or not.

Soft Clustering,

where each element has likelihood of belonging to a certain cluster.

Cluster analysis methods can be applied to a wide range of subjects. Basically it can be used in any context where finding groups in sets of data is useful, for example:

Image segmentation,

dividing an image into clusters or, more appropriate on this case, regions enhances a number of computer vision methods. Some examples are border detection or object recognition. [?]

Market analysis,

grouping enterprises [**?**] or consumers [**?**] to perform better market analysis or custom ads for each kind of consumer.

Education tracking,

grouping students to keep track of their record and apply more custom techniques to each student needs. [**?**]

Mathematical chemistry,

to analyse, group and find structural similarities in chemistry compounds, minerals, and any kind of material for which a chemical analysis is convenient. [**?**]

9.5 Stakeholders

The implied stakeholders here are, primarily the COMPSs developer team and, second, the rest of the research teams implied on GRID and cluster execution platforms interested on Cluster Analysis. They are the ones who will mostly benefit from this project. As stated earlier, this project wants to develop and explore the COMPSs framework. With this the programming model will be refined and, thanks to it's usage, more desired features can be found. Also the future usage of pyProCT will help to "advertise" and enhance it's possible diffusion, which is the main goal to keep this kind of programming models and frameworks, not just alive, but on a develop and improvement route.

On the other hand the pyProCT software was originally intended for protein clustering. However, the program core is quite generic. Thanks to the implementation of new plug-ins and modules to load, transform and use other kind of data inputs, the software is expected to suit a large group of researchers without too much work. Providing support for other usages is not part of this project but a possible speed up and performance enhancement will help them all and so, they are considered the main clients of the software itself and they will benefit from it.

9.6 State of the Art

Most of the clustering analysis methods are not new, however with the dramatical increase in data size mentioned earlier, researchers have focused on improving their performance as much as possible. From this need arise new, but more rough, methods such as **canonpy clustering** [?] which can process huge amounts of information but it just a pre-partitions data to then analyse smaller partitions with slower methods.

The increasing amount of information each data point contains it's also a problem for some algorithms. This information leads to high-dimensional data which, in turn, causes problems to a big part of the modern algorithms. This is known as the curse of dimensionality, which basically points out the fact that high-dimensional data often becomes sparse due to the large volume of space. It is important to note that this problem is not due to data itself but to the algorithm used. Some modern approaches try to overcome this difficulty by aiming to reduce the data-dimensionality, with methods such as **principal component analysis** [?], use just some part of it, like in **subspace clustering** [?] which have adopted ideas from density-based algorithms.

Apart from the clustering analysis techniques the focus of the project is the COMPSs refactoring. The election of the framework has been made according to two reasons: one, the proximity of the BSC research team which will ease the development, two, and most important, the framework is aimed to distributed computing on which the CA tasks are best executed.

Bibliography

[1] Jake Vanderplas. Why Python is Slow: Looking Under the Hood, May 2014.