UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) - BarcelonaTech
FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)

Integration of a parallel efficiency monitoring tool into an HPC production system

Computer Engineering speciality - Grau en Enginyeria Informàtica

---

Bachelor thesis

Author:
Helena Vela Beltran

Co-directors:
David Vicente Dorca (BSC), Marta García Gasulla (BSC)

Ponent:
Víctor López Herrero (BSC - AC department)

GEP Tutor:
Paola Lorenza Pinto

January 24, 2023

Table of Contents

1 Context and scope **9**
1.1 Introduction 9
1.2 Terms and concepts 9
1.2.1 High-Performance Computing (HPC) 9
1.2.2 Programming model 10
1.2.3 Message Passing Interface (MPI) 10
1.2.4 Performance Application Programming Interface (PAPI) 11
1.2.5 TALP 11
1.2.6 UserPortal 11
1.3 Motivation 13
1.3.1 Problem to be resolved 13
1.3.2 Justification 13

2 Project planning **13**
2.1 Scope 14
2.1.1 Requirements 14
2.1.2 Objectives 14
2.2 Stakeholders 16
2.2.1 Marenostrum Users 16
2.2.2 Operations department 16
2.2.3 Scientific community 16
2.2.4 University students 16
2.3 Methodology and rigour 16
2.3.1 Kanban 16
2.3.2 Git 17
2.3.3 Test environment 18
2.3.4 CI/CD 18
2.3.5 Group meetings 18
2.4 Tasks and planning 19
2.4.1 Project definition 19
2.4.2 Project planning 19
2.4.3 Project development 19
2.4.4 Project documentation 21
2.4.5 Estimates and the Gantt 23
2.5 Resources 24
2.5.1 Human resources 24
2.5.2 Hardware resources 24
2.5.3 Software resources 24
2.5.4 Material resources 25
2.6 Budget - Identification of costs 25
2.6.1 Human costs 26
2.6.2 Material costs 27
2.6.3 Other costs 29

2.6.4 Total cost 30
2.7 Risk management: alternative plans and obstacles 30
2.8 Management control 31
2.9 Sustainability 32
2.9.1 Self-assessment 32
2.9.2 Economical dimension 32
2.9.3 Environmental dimension 33
2.9.4 Social dimension 33

3 UserPortal's integration 34
3.1 Actual system design 34
3.2 Design decisions 35
3.3 Requirements 36
3.4 Backend development 36
3.4.1 Wrapper 36
3.4.2 Gathering Script 39
3.4.3 Parsing-Injecting Script 39
3.4.4 Module and cron 40
3.5 Database Adaptation 40
3.6 Web development 41
3.7 CI/CD 42
3.8 Testing 43

4 TALP - PAPI extension 44
4.1 Understanding how TALP works 44
4.2 Autoconf 46
4.3 Implementation 46
4.4 Testing 48
4.4.1 Data correctness and validation 49
4.5 Overhead evaluation 49

5 Making the new information meaningful 52

6 Future development 52

7 Conclusions 53

A Glossary 54

B GANTT diagram 56

C TALP execution diagram 57

D TALP execution diagram with PAPI 58

List of Figures

1 Visual reference of point-to-point inter-processor communication. Source: https://hpc.nmsu.edu/discovery/m
2 UserPortal main page, where all the user's jobs are listed. Users can also search and filter them. Source: Own . . . 12
3 UserPortal's job detail page. Even though the screenshot only reaches to CPU usage graph, various metrics are displayed, including memory usage, and power monitoring graphs. Source: Own . . . 12
4 Kanboard page for my user. Source: Own . . . 17
5 Usual git branching workflow. Source: https://www.nobledesktop.com/learn/git/git-branches 18
6 Diagram summing up each machine service execution. Source: Own . . . 35
7 Location and moving pipeline of TALP-generated files. Source: own . . . 38
8 Jobs and PopMetrics table in the UserPortal database. Source: DataGrip-generated . . . 41
9 Screenshot of a job webpage where it shows the TALP report with the POP Metrics. Source: Own . . . 42
10 Screenshot of a job webpage where it shows the TALP report with the POP Metrics and more than one monitoring region. Source: own . . . 42
11 Visual representation of connections made by GitLab runners. Source: own . . . 43
12 DLB-TALP own data structures. Source: own . . . 44
13 TALP workflow simplified chart. Source: own . . . 45
14 TALP workflow simplified chart with PAPI integration. Source: own . . . 48
15 LINPACK performance in GFLOPs in different scenarios and number of nodes used in the execution. Source: own . . . 50
16 PILS execution time in seconds in different scenarios and number of nodes used in the execution. Source: own . . . 51
17 HPCG performance in GFLOPs in seconds in different scenarios and number of nodes used in the execution. Source: own . . . 51
18 Gantt diagram for my project. Source: own . . . 56
19 TALP execution diagram I have made for myself to have a better understanding of the application. Source: own . . . 57
20 TALP execution diagram from before with the altered functions marked with purple Source: own . . . 58

List of Tables

1 Tasks hour estimation and resources needed. Source: own . . . 23
2 Estimated cost per role, price per hour is calculated taking into account that the BSC agreement stipulates 1792 working hours per year. Source: Own calculations and BSC agreement . . . 26
3 Table summarizing task cost estimation. Source: Own calculations . . . 27
4 Table with amortization costs. *MareNostrum4's amortization has been divided by 3456, the total number of nodes, because I am only using a node or even less during the development stages. Source: Own calculations . . . 28
5 Total cost of the project. Source: Own calculations . . . 30

6 Summary of the key concepts in the design 36

Abstract

This thesis presents the design, implementation, and evaluation of an extension of a library called TALP for tracing useful computation and performance metrics in MPI-based applications (Message Passing Interface). The extension also integrates the tool with a web portal called UserPortal. They are both developed at the Barcelona Supercomputing Center (BSC). The library captures information about communication patterns and computation performed by MPI applications, and makes this information available to users. The extension developed in this project adds the functionality of reading PAPI (Performance Application Programming Interface) counters, allowing users to know the instructions per cycle of their application and identify bottlenecks in their code.

UserPortal provides an easy-to-use interface for visualizing and analyzing the captured information and allows users to easily monitor the status of their jobs, such as memory usage, CPU usage, and their evolution over time. The integration of the library with the BSC system involves several stages of design and development, including a software wrapper, a modulefile, scripts for retrieving and processing data, and web development to display the data on the UserPortal.

However, users must be educated in performance analysis in order to effectively make a good reading and interpretation of the reported metrics and optimize their codes. A public documentation has been developed as well as a reference on how to use these tools on BSC machines, along with links to educational resources on related topics. Overall, this work provides a valuable tool for developers and researchers working with MPI-based applications, making performance optimization more approachable and efficient.

Resumen

Esta tesis presenta el diseño, implementación y evaluación de una extensión de una libreria llamada TALP para rastrear cálculos útiles y métricas de rendimiento en aplicaciones basadas en MPI ((*Message Programming Interface* por sus siglas en inglés en inglés). La extensión también integra la herramienta con un portal web llamado UserPortal. Ambos son desarrollados en el Barcelona Supercomputing Center (BSC). La librería captura información sobre patrones de comunicación y cálculo realizados por aplicaciones MPI y hace esta información disponible para los usuarios. La extensión desarrollada en este proyecto agrega la funcionalidad de leer los contadores PAPI (Performance Application Programming Interface), permitiendo a los usuarios conocer las instrucciones por ciclo de su aplicación e identificar cuellos de botella en su código.

UserPortal ofrece una interfaz fácil de usar para visualizar y analizar la información capturada y permite a los usuarios monitorizar fácilmente el estado de sus trabajos, como el uso de la memoria, el uso de CPU y su evolución en el tiempo. La integración de la librería con el sistema del BSC implica varias etapas de diseño y desarrollo, incluyendo un wrapper de software, un modulefile, scripts para recuperar y procesar datos y desarrollo web para mostrar los datos en UserPortal.

Aún así, los usuarios deben ser educados en análisis de rendimiento para poder hacer una buena lectura e interpretación de las métricas. Una documentación pública se ha desarrollado también para tener una referencia sobre como utilizar estas herramientas en las máquinas del BSC, junto con links a páginas que ofrecen recursos educacionales en temas relacionados. Finalmente, este proyecto provee una herramienta valiosa para desarrolladores e investigadores trabajando en aplicaciones basadas en MPI, haciendo la optimización de rendimiento más accesible y eficiente.

Resum

Aquesta tesi presenta el disseny, la implementació i l'avaluació d'una extensió d'una llibreria anomenada TALP per traçar el càlcul útil i les mètriques de rendiment en aplicacions basades en MPI (*Message Programming Interface* por les seves sigles en anglès). L'extensió també integra l'eina amb un portal web anomenat UserPortal. Tots dos són desenvolupats al Barcelona Supercomputing Center (BSC). La llibreria captura informació sobre patrons de comunicació i càlcul realitzat per aplicacions MPI i posa aquesta informació a disposició dels usuaris. L'extensió desenvolupada en aquest projecte afegeix la funcionalitat de llegir comptadors PAPI (Performance Application Programming Interface), permetent als usuaris saber les instruccions per cicle de la seva aplicació i identificar colls d'ampolla en el seu codi.

UserPortal proporciona una interfície fàcil d'utilitzar per visualitzar i analitzar la informació capturada i permet als usuaris monitoritzar fàcilment l'estat dels seus treballs, com ara l'ús de la memòria, l'ús de la CPU i la seva evolució en el temps. La integració de la llibreria amb el sistema del BSC implica diverses etapes de disseny i desenvolupament, incloent un envolupador de programari, un modulefile, scripts per recuperar i processar dades i desenvolupament web per mostrar les dades en el UserPortal.

No obstant, els usuaris han de ser educats en l'anàlisi de rendiment per tal de fer una lectura i interpretació efectiva de les mètriques reportades i ser capaços d'optimitzar els seus codis. S'ha desenvolupat una documentació pública així com una referència sobre com utilitzar aquestes eines en les màquines BSC, juntament amb enllaços a recursos educatius sobre temes relacionats. En general, aquest treball proporciona una eina valuosa per als desenvolupadors i investigadors que treballen amb aplicacions basades en MPI, fent que l'optimització del rendiment sigui més accessible i eficient.

Acknowledgement

I cannot express enough gratitude to my co-directors and ponent for their unwavering support and guidance throughout the conception, design, and implementation of this project. Their expertise is invaluable and I am deeply grateful for the opportunity to work with such talented people.

I am also deeply grateful to the User Support Team for their constant support and motivation since the beginning of my journey, long before this project even existed. I am truly blessed to have such a supportive and loving team, who I consider to be my own teeny tiny family. My heartfelt thanks also go out to my friends within the BePPP group, who have brought me so much joy and support throughout this years.

I am filled with gratitude for all the amazing people I have met during my undergraduate studies and the impact they have had on my life. None of this would have been possible without the friendships and camaraderie I have formed inside and outside the classroom, and the Omega building on campus will always hold a special place in my heart.

Special thanks to my beloved significant other for all the amazing dinners he made to me when I was drowning in work.

1. Context and scope

1.1. Introduction

This project has been born in my current workplace, as a collaboration of me, a member of the User Support at the Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC-CNS) with the Best Practices for Performance and Programmability group (BePPP), in the Computer Sciences department inside BSC-CNS as well.

My group, the User Support Team inside the Operations department, led by David Vicente, focuses on helping and managing the users of the BSC clusters with their issues and guaranteeing an appropriate and efficient usage of the resources. Our main tasks include solving the request of researchers using the BSC HPC-Resources, installation and debugging of applications, enabling and porting codes to the MareNostrum architecture, optimization and scalability studies, parallelization assistance, benchmarking of applications, manage accounting information and users accounts.

The BePPP group, led by Marta García, quoting their team webpage [1]: "[...] Aims to be the bridge between scientific domain researchers and computer scientists researchers. Promoting best practices for programmers to productively (re)structure their codes in ways that can result in high efficiency and portability".

They achieve that by promoting the use of BSC tools and methodologies, providing hints on how applications performance can be improved through analysis or proof-of-concept studies, also on improving programming models to address real applications issues/needs, and much more.

The project is to integrate the library developed and maintained by BePPP group, TALP, into the User Support Team web page UserPortal. Also, extending its functionalities to obtain hardware counter metrics as well.

1.2. Terms and concepts

In this section, I will briefly describe and explain some of the terms and concepts necessary to understand the context of this project. Also, a Glossary is added in the Appendix (A) for short definitions.

1.2.1. High-Performance Computing (HPC)

High-Performance Computing can be defined as "the ability to process data and perform complex calculations at high speeds" [5]. Supercomputers are developed to perform this task. Supercomputers work as a large number of computers executing parallel code, the most powerful ones reaching a quintillion of

[1]https://www.bsc.es/discover-bsc/organisation/research-departments/best-practices-performance-and-programmability

calculations per second [3]. In the context of this project, we will focus on MareNostrum4 [2], the most important supercomputer at the BSC. Even though the project can and it will be adapted to other BSC machines easily, the first stages of the development and tests will be performed in that cluster.

Normally, we refer to a program execution as a ”job”. As you do not execute the program directly, but send it to a batch queue where the queue manager puts it in execution as soon as the resources are available.

1.2.2. Programming model

A parallel programming model is a layer that isolates the programmer from the parallel architecture, the same way the operating system abstracts the user from the hardware architecture.

A parallel programming model allows the developer to exploit parallelism within its code abstracting it from the hardware features.

There are an extensive number of parallel programming models, but we can divide them into two main categories depending on the hardware platform they target: Distributed memory and shared memory parallelism.

The most widely used are respectively: MPI and OpenMP, we focus on MPI because is the one that TALP traces to detect load imbalances and excessive communication times.

In the ecosystem of parallel software in which we move, a programming model is often called using an API to expose hardware features and exploit this to achieve high performance. Many parallel architectures exist, so it would be impractical to have a programming language for each of them. Examples of programming models in the BSC are DLB [3] and COMPSs [4].

1.2.3. Message Passing Interface (MPI)

MPI is a programming model based on a standard API definition. [7]. It has several implementations, some open-source as OpenMPI [9] and MPICH [8], or even proprietary ones like IntelMPI. It is used in almost all the used programming languages and applications as well, and it is the main standard used in HPC.

In general terms, the user explicits the communication between a set of processes from an API that implements an MPI library. This includes sbarrier operations (synchronizations, ..), point-to-point operations (synchronous and asynchronous), collective operations (gather, reduce, etc.). ll of this maintaining a coherent state among processes.

[2]https://www.bsc.es/es/marenostrum/marenostrum
[3]https://www.bsc.es/research-and-development/software-and-apps/software-list/dlb-library-dynamic-load-balancing
[4]https://www.bsc.es/research-and-development/software-and-apps/software-list/comp-superscalar/

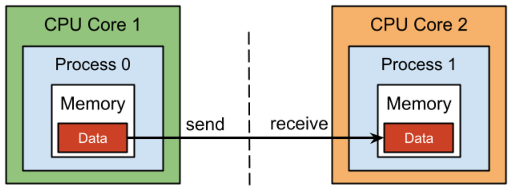


Figure 1
Visual reference of point-to-point inter-processor communication.
Source: https://hpc.nmsu.edu/discovery/mpi/introduction/

1.2.4. Performance Application Programming Interface (PAPI)

PAPI [5] is a programming interface to see processor events. Events are occurrences of specific signals related to a processor's function. Hardware performance counters exist as a small set of registers that count events, such as cache misses and floating point operations while the program executes on the processor. These events are dependent on the processor architecture.

Quoting their documentation [4]: "PAPI provides [...] a consistent interface and methodology for the use of low-level performance counter hardware found across the entire computing system (i.e. CPUs, GPUs, on/off-chip memory, interconnects, I/O system, energy/power, etc.). PAPI enables users to see, in near real time, the relations between software performance and hardware events across the entire computer system." Both hardware and software performance events are monitored through one interface.

1.2.5. TALP

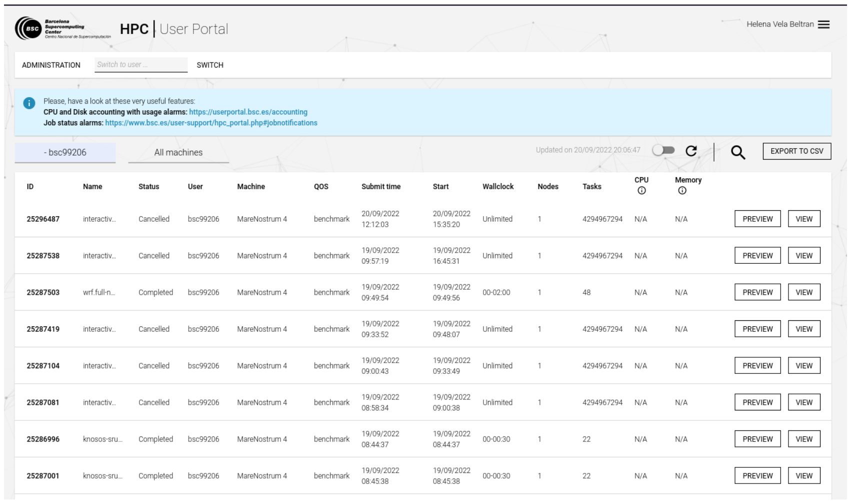
TALP, the tool we will integrate and expand, is a part of the DLB project, developed by the BePPP group. TALP is a "lightweight, portable, extensible, and scalable tool for online parallel performance measurement"[6]. It reports metrics to better understand the parallel efficiency of an application, by analyzing an MPI region and seeing if the execution time is meaningful or if too much time is lost in MPI communication. It can be linked against the program or use an API to define different regions and analyze them at runtime. It generates a report with the collected metrics, it can also collect metrics per node, per process, or all at once. It is lightweight as it barely introduces overhead to the applications' execution time.

1.2.6. UserPortal

UserPortal [6]is a Web Portal developed and maintained by the User Support Team since 2018, in which users with access can take easily a look at their jobs and their metrics in a unified way, regardless of

[5]https://icl.utk.edu/papi/
[6]https://userportal.bsc.es/

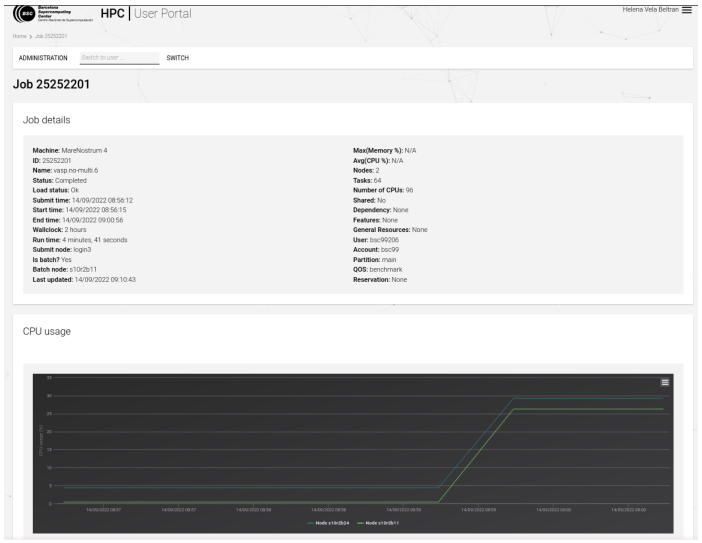
whether they have various accounts or different allocations in more than one cluster. Also, the nodes' state and the clusters' queues are shown.



Please, have a look at these very useful features:
CPU and Disk accounting with usage alarms: https://userportal.bsc.es/accounting
Job status alarms: https://www.bsc.es/user-support/hpc_portal.php#jobnotifications

ID	Name	Status	User	Machine	QOS	Submit time	Start	Wallclock	Nodes	Tasks	CPU	Memory		
25296487	interactiv...	Cancelled	bsc99206	MareNostrum 4	benchmark	20/09/2022 12:12:03	20/09/2022 15:35:20	Unlimited	1	4294967294	N/A	N/A	PREVIEW	VIEW
25287538	interactiv...	Cancelled	bsc99206	MareNostrum 4	benchmark	19/09/2022 09:57:19	19/09/2022 16:45:31	Unlimited	1	4294967294	N/A	N/A	PREVIEW	VIEW
25287503	wrf.full-n...	Completed	bsc99206	MareNostrum 4	benchmark	19/09/2022 09:49:54	19/09/2022 09:49:56	00-02:00	1	48	N/A	N/A	PREVIEW	VIEW
25287419	interactiv...	Cancelled	bsc99206	MareNostrum 4	benchmark	19/09/2022 09:33:52	19/09/2022 09:48:07	Unlimited	1	4294967294	N/A	N/A	PREVIEW	VIEW
25287104	interactiv...	Cancelled	bsc99206	MareNostrum 4	benchmark	19/09/2022 09:00:43	19/09/2022 09:33:49	Unlimited	1	4294967294	N/A	N/A	PREVIEW	VIEW
25287081	interactiv...	Cancelled	bsc99206	MareNostrum 4	benchmark	19/09/2022 08:58:34	19/09/2022 09:00:38	Unlimited	1	4294967294	N/A	N/A	PREVIEW	VIEW
25286996	knosos-sru...	Completed	bsc99206	MareNostrum 4	benchmark	19/09/2022 08:44:37	19/09/2022 08:44:37	00-00:30	1	22	N/A	N/A	PREVIEW	VIEW
25287001	knosos-sru...	Completed	bsc99206	MareNostrum 4	benchmark	19/09/2022 08:45:38	19/09/2022 08:45:38	00-00:30	1	22	N/A	N/A	PREVIEW	VIEW

Figure 2
UserPortal main page, where all the user's jobs are listed. Users can also search and filter them.
Source: Own



Job 25252201

Job details

Machine: MareNostrum 4
ID: 25252201
Name: vasp.no-multi.6
Status: Completed
Load status: Ok
Submit time: 14/09/2022 08:56:12
Start time: 14/09/2022 08:56:15
End time: 14/09/2022 09:00:56
Wallclock: 2 hours
Run time: 4 minutes, 41 seconds
Submit node: login3
Is batch? Yes
Batch node: s10r2b11
Last updated: 14/09/2022 09:10:43

Max(Memory %): N/A
Avg(CPU %): N/A
Nodes: 2
Tasks: 64
Number of CPUs: 96
Shared: No
Dependency: None
Features: None
General Resources: None
User: bsc99206
Account: bsc99
Partition: main
QOS: benchmark
Reservation: None

CPU usage

Figure 3
UserPortal's job detail page. Even though the screenshot only reaches to CPU usage graph, various metrics are displayed, including memory usage, and power monitoring graphs.
Source: Own

1.3. Motivation

1.3.1. Problem to be resolved

HPC is expensive. The cost of each machine reaches up to millions of euros[1], usually coming from public funding (at least in BSC's case[2]), having to cover not only the hardware, but the resources needed and the operations team in charge to maintain it. Poorly programmed parallel code, which is more frequent than you would expect in scientific applications since parallel programming requires technical knowledge and training, leads to loss of computing power and a waste of resources, usually cores in an idle state, consuming electricity and blocking other users from executing their programs in them. Performance analysis tools are developed in order to help programmers identify issues with their codes. Still, most of the time these are not friendly at all for people without a computing or hardware background because of the metrics they represent and the usability of these softwares.

1.3.2. Justification

The aim of this project is to make performance analysis with TALP more easy and reachable for researchers without a programming or computing background. Also, to expand it to obtain other meaningful metrics from hardware counters. This way, better usage of the cluster and a less waste of resources (and in consequence, money) will be gradually obtained.

Since the two parts (integration and expansion) of this project have been born and it will be developed entirely in the BSC environment, considering the particular architectural traits of its infrastructure, and it will not include third parties, no other previous similar problems or solutions exist in the BSC scope.

User Portal was developed in 2018 and its constant expansion to make BSC's researchers' life easier embraced this collaboration with the Computer Sciences department, its development has been always a User Support Teams task. Some functionalities of it have been improved over time (like the accuracy of the CPU load, to mention one), but an integration of this kind has never been performed. It is also the first time working with hardware counters inside TALP since these might introduce extra overhead to the execution.

2. Project planning

The project should take approximately 540 hours. It starts in September 2022 and should end by January of 2023. By week, part of my work hours and extra off-work hours are dedicated to this, on average, 30 hours per week are solely committed to the project.

The task description and the planning of these will be covered in this section. In general terms, we can divide the tasks into general sections. We will have a first "Project conception" task, to see if this project is feasible. Then, comes the "Project planning" itself, after that, the "Project development", and finally, "Project documentation". Even though I mentioned these in order, they might not have been executed in that particular order, even though they might overlap in time. These general tasks are divided into smaller sub-tasks.

2.1. Scope

The objectives and sub-objectives will be defined in this section, as well as requirements, possible risks, and obstacles.

2.1.1. Requirements

Before starting to define the project, we need to make clear its requirements. These are what the project must have by the end of the development. As explained, the main objective is to integrate TALP into UserPortal, and add PAPI counters into TALP to calculate the applications' IPC. Thus, the requirements are:

- Make TALP usage transparent for MareNostrum 4 users. This way is as easy as loading some environment to use it.
- Be able to see TALP reports from UserPortal.
- Find the calculated IPC in TALP's report. Also raw metrics as the traced number of cycles and instructions, respectively.
- Write documentation for users to be educated in performance analysis so they can understand the reports and improve their codes.

2.1.2. Objectives

For this project, the main objectives are established for each of the two parts. Some extras have been defined having in mind further development and continuity, but since the limited time of this project, it might be out of the scope.

The objectives and steps for each part are:

- **Integration with UserPortal.** These steps have not necessarily been followed in this particular order.
  - Understand the UserPortal backend. UserPortal backend is actually a system called Monitoring Service, developed entirely by Carlos Tripiana, a former User Support Team member. It generates, transports, process, and inserts in our databases files containing information about the jobs in our clusters, regardless of their state.

- Test and investigate how TALP works and how it is invoked when launching an application that we want to trace.
- Development and testing of a command acting as a wrapper for launching applications with TALP tracing enabled. Since the idea is to make this as easy and transparent as possible for the user, it will be launched with the mpirun command, present in the Intel implementation of MPI [7] and the OpenMPI one [8], and it will work exactly the same as them, but making all the exports and setting all the configurations required for enabling TALP and saving the report file.
- Development and testing of a script in charge of fetching all the TALP files stored in the clusters, bringing them to the VM where UserPortal runs, and deleting them from the original location to not overload the filesystem with redundant information.
- Design and creation of a new table in the UserPortal database, with all the necessary columns and restrictions to fulfil the integrity of the possibly fetched TALP metrics.
- Development and testing of a script in charge of parsing, processing, and inserting data to the previously mentioned table. Having in mind all types of error control to reassure the whole database's security and integrity.
- Understand the UserPortal frontend with all of its layers, especially how data is retrieved from the database and displayed in the web portal.
- Add all the necessary code to the UserPortal frontend to display the desired data.
- Test the whole system to detect and correct possible bugs, failures, and areas to improve.

• **TALP expansion to collect hardware counters metrics.** Since I am not familiar with TALP's implementation or PAPI, the specification of the tasks is more generic.

- Agree with group leaders on which data we want to trace exactly.
- Explore PAPI and how it is used.
- Understand TALP's inner implementation and how it works.
- Investigate and implement the usage of hardware counters inside TALP.
- Test the correctness and good performance in terms of reliability of the new implementation.
- Perform benchmarks to determine how much overhead is introduced with the usage of hardware counters.

[7]https://www.intel.com/content/www/us/en/develop/documentation/mpi-developer-reference-linux/top/command-reference/mpirun.html
[8]https://www.open-mpi.org/doc/current/man1/mpirun.1.php

2.2. Stakeholders

The following communities could benefit from this project.

2.2.1. Marenostrum Users

One of the main purposes of the project is to make using TALP easier and almost transparent for the users, therefore they will be more encouraged to use performance analysis tools to trace their applications. Thus, the BSC researchers are the stakeholders for who we designed the integration part.

2.2.2. Operations department

If executions consume less computation time, the Operations Department will be able to offer more compute hours to users who need it. Thus, more projects could be allocated in MareNostrum 4.

2.2.3. Scientific community

The TALP expansion will benefit people from the scientific community (mainly developers of scientific applications and computer scientists), who want to obtain hardware counter metrics from their application execution in addition to the parallel efficiency, so they can identify better the issues within their codes.

2.2.4. University students

As TALP is under a GPL license, students who want to investigate and start to introduce themselves to the world of performance analysis and parallel computing will be able to use this tool and, as the scientific community, start to trace applications for a better understanding of them.

2.3. Methodology and rigour

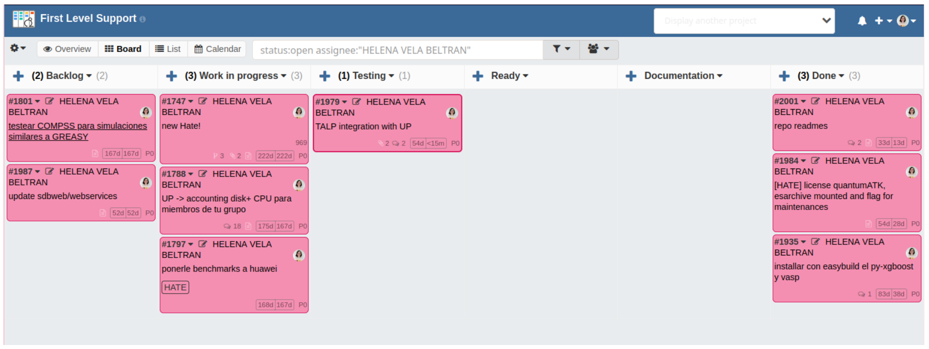
This section will explain all the tools that we used to track the management and the process of the project, and also for putting into work a best practices workflow for programmers.

2.3.1. Kanban

Internally, the User Support Team uses the *Kanban* agile methodology for tracking week to week our tasks. It consists of a visual board, containing various columns and tasks displayed as cards. Each column represents the task's state, in our case, the work is divided into these 6 stages:

- Backlog - for tasks that have not been started or do not even have an owner.
- Work in progress - for tasks with an owner who is working on it.
- Testing - when the main development is made and in the testing phase.
- Ready - for tasks that passed all the tests and are ready to be deployed in a production system.
- Documentation - the documentation process for that task.
- Done - completed tasks.

This method provides an easy-to-understand way to track your tasks and perfect visualization of the worker's workflow, reduces waste and inefficiency, and increases team focus by limiting work in progress. It is different from Scrum in the way that Kanban is more flexible, and is not based on sprints, ceremonies, and roles.[10] In our team, we use Kanboard [9], which is a free and open-source Kanban project management software.



status:open assignee:"HELENA VELA BELTRAN"

#1801 HELENA VELA BELTRAN
testear COMPSS para simulaciones similares a GREASY
167d 167d P0

#1987 HELENA VELA BELTRAN
update sdbweb/webservices
52d 52d P0

#1747 HELENA VELA BELTRAN
new Hate!
969
3 2 222d 222d P0

#1788 HELENA VELA BELTRAN
UP -> accounting disk+ CPU para miembros de tu grupo
18 175d 167d P0

#1797 HELENA VELA BELTRAN
ponerle benchmarks a huawei
HATE
168d 167d P0

#1979 HELENA VELA BELTRAN
TALP integration with UP
2 2 54d <15m P0

#2001 HELENA VELA BELTRAN
repo readmes
2 33d 13d P0

#1984 HELENA VELA BELTRAN
[HATE] license quantumATK, esarchive mounted and flag for maintenances
54d 28d P0

#1935 HELENA VELA BELTRAN
installar con easybuild el py-xgboost y vasp
1 83d 38d P0

Figure 4
Kanboard page for my user. Source: Own

Inside each card, you can add comments, pictures, sub-tasks, documents, and much more to keep a better track of what are you doing and relevant information you might need, also for other team members interested in seeing what are you doing or in helping you.

2.3.2. Git

Git [10] is a well-known open-source software for distributed version control. When collaborating on a big project, it is easy to mess up your work with your companions if any kind of control nor communication is present. At BSC (both the Support team and BePPP group) we use Git for better control and management of our repositories. In the Support team, we have our own private GitLab domain for internal projects, and the BePPP group shares DLB under a GitHub repository [11].

[9]https://kanboard.org/
[10]https://git-scm.com/
[11]https://github.com/bsc-pm/dlb

The usual workflow is that for each new feature, a development branch is created, thus we can be sure that any production environment has not been compromised until the development has finished and all the required tests have been performed. Once we are sure that the feature is ready to be in the production system, the development branch is merged with the main one.

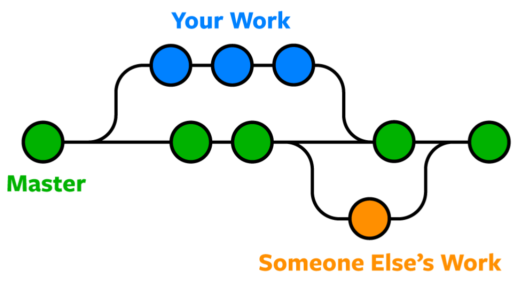


Figure 5
Usual git branching workflow.
Source: https://www.nobledesktop.com/learn/git/git-branches

2.3.3. Test environment

In addition to Git for version control and testing, in the User Support Team, we usually have a separate environment, identical to the production one to test repositories with a webpage involved, like UserPortal. Inside the same VM, two separate repositories, one for test and the production one, are hosted and point to different URLs (userportal-test.bsc.es in this case), this way we can experiment with the user interface with total freedom and see the results instantly.

Once the development has finished, the test repository is merged into the production one.

2.3.4. CI/CD

Both the BePPP and the Operations Team, make use of CI/CD (see definition in the Glossary A). On one hand, to perform tests to ensure that with each commit everything still works as expected, and on the other, to make deployments easier for the simultaneous environments.

2.3.5. Group meetings

Every two weeks, a group meeting is scheduled. In this one, we meet me and David Vicente (User Support) with Marta Garcia-Gasulla and Víctor López (BePPP) to check my updates on the project. We also discuss improvements that can be done or design features for the upcoming tasks, also as the next steps. For the more User Support-exclusive topics, I normally consult directly with David, or I comment it in the User Support Team weekly meetings, which are more generic to the whole group projects and tasks.

2.4. Tasks and planning

During the following sub-sections, the tasks of the project are described.

2.4.1. Project definition

The objective of this phase is to study and determine the main requirements of the system.

- **PI1 - System viability:** In this stage, a preliminary study had to be made to determine if the project was feasible and useful. Here we analyzed the main goal and different ways to implement it, and the best way to do it was decided as well. This took about 1 hour with the project directors. Since it is the prerequisite to all the tasks in the project, this is omitted in the table.

2.4.2. Project planning

A big part of the project is about planning the project itself, only the team members, a computer, and maybe a notebook to take notes.

- **PP1 - Context and Scope of the project:** A study to define the context and the scope is performed. This takes about 25 hours and the text processor Overleaf. [12]
- **PP2 - Planning:** Once we have the PP1, tasks need to be specified and scheduled, with a Gantt diagram for better visualization. Also resources specification. This takes about 20 hours, and again, Overleaf is the text editor.
- **PP3 - Budget and sustainability:** After the tasks are concertized, a budget will be provided considering the number of tasks and resources needed. This will take 25 hours and working with Overleaf as the text editor.
- **PP4 - Integration in final document:** Once we have all the previous documents, they will be joined in a project definition file. This takes up to 15 hours considering modifications that may need to be made. Again, the Overleaf text editor will be used.
- **PP5 - Biweekly meetings:** An hour every two weeks is dedicated to joining the two groups and talking about the project's progress. This makes 12 hours in average.

2.4.3. Project development

Most of these tasks were already explained in the objectives part of the previous deliverable. These form the core of the project.

User Support Tasks

[12]https://www.overleaf.com

- **US1 - System concretization** Study existing structures (Database, filesystems, UserPortal's structure) and see how the new data will be fitted into it. I include the understanding of TALP in this section as well to learn about the metrics it produces and how they will fit in our system. 20 hours will be needed for this since the coherence of the system must respect the present structures and these have to be discussed with the directors.
- **US2 - Command development** Develop a new command for activating TALP, loading the required modules, and making the necessary exports. Also configuring the metrics to collect and the output file. This will take about 10 hours because bash scripting needs to be recalled since I do not have any experience, and will require Vim [13] as the text editor.
- **US3 - Database adaptation** Creating the new table with the required rows where the collected metrics will be placed. This task will take about 15 hours because I am not familiar with the technology used for UserPortal's databases. The software Datagrip connects to and modifies the database.
- **US4 - Development of a script for retrieving and deleting files** This script will fetch and gather the TALP generated files from the /tmp filesystems across the machine, following a particular pattern for the name in order to recognize them. Once gathered, they will be deleted not to fill the filesystem with redundant information. This will take about 15 hours, and only the text editor Vim will be needed.
- **US5 - Development of a module for users to use the command** A module is created, acting as a wrapper of the wrapper command, exporting the necessary paths for users to use the developed command. This will take an hour approximately and only Vim again is required as a text editor.
- **US6 - Development of a JSON parser script** For parsing the TALP generated files and inserting the collected metrics into UserPortal's databases. This takes about 25 hours and Vim as the used text editor. As well as Datagrip to check the inserted data into the databases.
- **US7 - Modify UserPortal's code to display the new information** Web development in all of UserPortal's layers to show the new data. PHPStorm's[14] software is needed to modify the code and this will take about 40 hours due to the complexity of the code and my inexperience in web development.
- **US8 - Test each of the components individually** Exhaustive test is needed to guarantee that all bugs will be corrected when all of this software is deployed into production and users start to use it. This will take about 40 hours.
- **US9 - Test the system as a whole** Same as the previous task but with the closest production environment possible. This will take 5 hours since mostly all bugs would have been corrected already.
- **US10 - Deploying everything into a production environment** Everything has been developed

[13]https://www.vim.org/
[14]https://www.jetbrains.com/es-es/phpstorm/

in a test environment and needs to be moved to the production webpage and database. This will take about two hours. PHPStorm and Datagrip software, and git to manage the repositories.

TALP Tasks

- **TT1 - Study TALP's code** TALP's particular structures and algorithms must be studied in order to make a good and respectful implementation of the hardware counters. This should take about 18 hours.
- **TT2 - Read documentation about PAPI** As a person who never used PAPI before, documentation and some practice must be absorbed. This will take about 16 hours.
- **TT3 - Study which counters to implement** Once I have knowledge about PAPI, we must see which counters better fit the code's purpose without perverting it too much since the idea is to keep it lightweight. This will take about 10 hours including discussions with directors about this decision.
- **TT4 - Implement PAPI counters support** Now the actual implementation starts. TALP's code will be modified to support PAPI counters being really careful about their activation to not introduce much overhead. This will take about 45 hours. Only Vim is needed as a text editor.
- **TT5 - Modify command line arguments** Since users may not always want to use PAPI counters, command line arguments when enabling TALP must be modified to alter their behaviour depending on the options. This should take about 10 hours depending on the counters implemented. Only Vim is needed as a text editor.
- **TT6 - Perform tests to detect bugs and errors** Once everything is implemented, executions to test the code robustness are required. This will take about 40 hours to be sure about everything is alright.
- **TT7 - Perform benchmarks to test introduced overhead** As soon as we can be sure that the implementation is solid and alright, benchmarks to test the overhead introduced by hardware counters tracing will be launched. This includes posterior analysis and maybe corrections on the code. This will take about 30 hours.
- **TT8 - Deploy into production environment** This code implementation and development will be made into a separate branch to not alter the production software. A delicate final merge in git will be performed, and the related public/private repositories and documentation must be updated. This will take about 6 hours and includes checking that everything is alright.

2.4.4. Project documentation

- **PD1 - Write support's internal documentation** Document inside User Supports internal documentation as an annex to UserPortal section. This will take about 5 hours and draw.io[15] as the

[15] https://app.diagrams.net/

software to generate charts to explain the new system.

- **PD2 - Write support's external documentation** For users to know how to use it, also integrating it in a new Performance Analysis new section inside our tools documentation. This will take about 10 hours and Vim as the text editor.
- **PD3 - Write documentation on the new TALP features for BePPP** Write the documentation about the newly implemented part inside TALP for CS people. This will take about 10 hours.
- **PD4 - Revise all documentation** Revision to check that all the documentation is alright. This will take about 3 hours.
- **PD5 - Write the project's technical report** Project's development technical report writing. This is a tedious task of about 50 hours and only needs Overleaf as the text processor.
- **PD6- Thesis preparation** The preparation of the presentation itself, will require about 20 hours having in mind that a presentation and a script must be crafted. Only Overleaf will be needed for this.

2.4.5. Estimates and the Gantt

ID	Name	Duration(h)	Predecessors	Resources
	Project inception	1		
PI1	System viability	1		
	Project planning	97		
PP1	Context and Scope of the project	25		Overleaf
PP2	Planning	20	PP1	Overleaf
PP3	Budget and sustainability	25	PP2	Overleaf
PP4	Integration in final document	15	PP3	Overleaf
PP5	Weekly meetings	12		
	Project development	351		
	User Support tasks	176		
US1	System concretization	20		Datagrip, draw.io
US2	Command development	10	US1	Vim
US3	Database adaptation	15		Datagrip
US4	Development of a script for retrieving and deleting files	15	US1	Vim
US5	Development of a module for users to use the command	1	US1	Vim
US6	Development of a json parser script	25	US1,US3	Vim, Datagrip
US7	Modify UserPortal's code to display the new information	40	US3, US6	PHPStorm
US8	Test each of the components individually	40	US2,US3,US4, US5,US6,US7	
US9	Test the system as a whole	5	US8	
US10	Deploy everything into a production environment	5	US9	PHPStorm, Datgrip, Git
	TALP Tasks	175		
TT1	Study TALP's code	18		
TT2	Read documentation about PAPI	16		
TT3	Study which counters to implement	10	TT2	
TT4	Implement PAPI counters support	45	TT3	Vim
TT5	Modify command line arguments	10	TT4	Vim
TT6	Perform test to detect bugs and errors	40	TT5	
TT7	Perform benchmarks to test introduced overhead	30	TT6	
TT8	Deploy into production environment	6	TT7	Git
	Project documentation	98		
PD1	Write support's internal documentation	5	US10	
PD2	Write support's external documentation	10	US10	Vim
PD3	Write TALP's new features documentation	10	TT8	
PD4	Revise all documentation	3	PD3	
PD5	Write the project's technical report	50		Overleaf
PD6	Thesis preparation	20	PD5	Overleaf
TOTAL		547		

Table 1
Tasks hour estimation and resources needed. Source: own

Because of the huge size of the Gantt diagram, it can be found in the Appendix. 18

2.5. Resources

Every project relies on a set of resources in order to be conceived, developed, tested, deployed, and documented. In this project, I categorize the resources needed into these four groups: Human resources, Hardware resources, Software resources, and Material resources.

2.5.1. Human resources

Mainly, 5 humans are involved in this project. First of all, I, the developer, will be in charge to bring the project to life. Then, the 2 co-Directors and the Ponent, will assist me in the decisions involved in the project and assess me during the development. Finally, my GEP tutor, will help me as well with the Project planning tasks during the first month of the project.

2.5.2. Hardware resources

The essential hardware resources for this project are computers. We differentiate two main computers involved and needed:

- My work laptop: Dell latitude 7490 [16]. This is from where I connect to the required clusters and/or servers, and also make development in my local machine.
- Marenostrum 4: The cluster from where all the User Support-related scripts will be developed having in mind the filesystem.

It is important to have into consideration the router which allows me to connect to the network and the physical machine where UserPortal is deployed, both for test and production environments.

2.5.3. Software resources

Various software is needed. For communication through me and the co-Directors+Ponent, we use Thunderbird [17] as the mail client, we also have a channel in RocketChat [18] for more informal and quick communications compared to email ones. And in case it is required, Zoom meetings [19] would be used for group meetings, but we normally meet in person. This software are enforced by the BSC to support the use of Open Source software. Also, lmod [20] is the software used for creating a module environment.

[16] https://www.dell.com/support/kbdoc/es-es/000132081/gu%C3%ADa-visual-de-dell-latitude-7490
[17] https://www.thunderbird.net/es-ES/
[18] https://www.rocket.chat/
[19] https://zoom.us/
[20] https://lmod.readthedocs.io/en/latest/

Git is used for version control, and programming languages as Python [21], PHP [22], C [23] and Bash [24] will form the project's technology base.

For documentation and text processing, I personally use Overleaf as my favorite text processor and Vim as my preferred text editor for scripting in the clusters. For this report, the Gantt chart generator [25] has been used.

I benefit from a free student license for the JetBrains [26]suite, which allows me to use PHPStorm for coding in big and complex PHP projects. Also, I enjoy Datagrip for database connection, consulting, and modification.

2.5.4. Material resources

This includes all sorts of documentation and papers that I devoured to gain the necessary knowledge in the making of the project.

2.6. Budget - Identification of costs

The elements mentioned in the following sub-sections are the ones being taken into account for the budget calculation. In addition, a management control mechanism is defined for surpassing deviations that might appear during the making of this project.

Calculating this project's expenses is a bit more difficult because I work from home for 2 of the 5 working days, so bills related to work that I have to pay are involved in the calculation of the costs. I estimate, for 5 months of project duration, approximately 22 weeks. Which make:

- Worked days in Barcelona = 22 weeks * 5 days a week * 3/5 days of the week worked in BCN = 66 days
- Worked days from home = 22 weeks * 5 days a week * 2/5 days of the week working from home = 44 days

That will be crucial to calculate the proportional part of a cost depending on the location. Also, I estimate around 4.5 daily hours of dedication to the project on average.

[21] https://www.python.org/
[22] https://www.php.net/manual/es/intro-whatis.php
[23] https://en.cppreference.com/w/c/language
[24] https://www.gnu.org/savannah-checkouts/gnu/bash/manual/bash.html
[25] https://www.onlinegantt.com/
[26] https://www.jetbrains.com/

2.6.1. Human costs

Even though it can sound a bit obnoxious, people is what makes this project possible. Human resources are needed in every step of it. We can differentiate 4 roles involved with tasks during the course of the project are:

- **Developer** - Programmer and tester of all the code. I will be the main and only developer in this project. This role is manifested inside our project in the tasks inside User Support tasks (US{1-10}) and TALP tasks (TT{1-8}, excluding TT7).
- **Technical writer** - Person who documents every existing phase of the project and also writes any kind of documentation or report. I will be again, the one and only person assuming this role. This role will be executed in all the PDX tasks of this project (PD1,PD2,PD3,PD4,PD5,PD6).
- **Researcher** - Role that will be in charge of experimenting with the code in order to analyze results and draw conclusions. I, as well, will entirely execute this role. This applies only for task TT7.
- **Project manager** - Responsible for the planning, supervision, and correct development of the project. We can find here the two co-directors of the project, the ponent, and the GEP tutor. This only applies for task PP5, but involves 4 people in it.

For calculating the cost of the tasks (Costs per Activity (CPA)), roles are assigned to each task. For each task, the total cost will be:

$$\textit{Task cost} = \textit{cost of the role} * \textit{assigned hours to task}$$

Estimated role costs, based on salary ranges of the BSC for students and Glassdoor for Project leader references, with the added 35% of the Social Security tribute, added to calculate the cost per year are:

Role	Gross Salary per year (€)	Cost per year (€)	Price per hour (€)	Played by
Developer	14.300	22.000	12,75	Me
Technical writer	14.300	22.000	12,75	Me
Researcher	14.300	22.000	12,75	Me
Project manager	32.500	50.000	27,90	GEP tutor, co-directors, and ponent

Table 2
Estimated cost per role, price per hour is calculated taking into account that the BSC agreement stipulates 1792 working hours per year. Source: Own calculations and BSC agreement

With this information, we can now calculate the cost per task:

ID	Name	Duration(h)	Hours				Total cost (€)
	Project inception	1	Developer	Technical writer	Researcher	Project manager	
PI1	System viability	1	1			1	12,76
	Project planning	97					
PP1	Context and Scope of the project	25		25			319
PP2	Planning	20		20			255,2
PP3	Budget and sustainability	25		25			319
PP4	Integration in final document	15		15			191,3
PP5	Weekly meetings	12	12			12 (x4 persons)	2.831,52
	Project development	351					
	User Support tasks	176					
US1	System concretization	20	20				255,2
US2	Command development	10	10				127,6
US3	Database adaptation	15	15				191,3
US4	Development of a script for retrieving and deleting files	15	15				191,3
US5	Development of a module for users to use the command	1	1				12,76
US6	Development of a json parser script	25	25				319
US7	Modify UserPortal's code to display the new information	40	40				510,4
US8	Test each of the components individually	40	40				510,4
US9	Test the system as a whole	5	5				63,8
US10	Deploy everything into a production environment	5	5				63,8
	TALP Tasks	175					
TT1	Study TALP's code	18	18				229,68
TT2	Read documentation about PAPI	16	16				204,16
TT3	Study which counters to implement	10	10				127,6
TT4	Implement PAPI counters support	45	45				574,2
TT5	Modify command line arguments	10	10				127,6
TT6	Perform test to detect bugs and errors	40	40				510,4
TT7	Perform benchmarks to test introduced overhead	30			30		382,8
TT8	Deploy into production environment	6	6				76,56
	Project documentation						
PD1	Write support's internal documentation	5		5			63,8
PD2	Write support's external documentation	10		10			127,6
PD3	Write TALP's new features documentation	10		10			127,6
PD4	Revise all documentation	3		3			63,8
PD5	Write the project's technical report	50		50			638
PD6	Thesis preparation	20		20			255,2
TOTAL		547					10,285.14

Table 3
Table summarizing task cost estimation. Source: Own calculations

2.6.2. Material costs

Software is not considered for the material costs because all the software I am using is free for me, either because it is Open Source or I have a free student license.

Amortization

The amortization of the material resources used in the project. This includes every material being that has been used. From hardware to my office furniture. The following table down below summarizes every amortization I have considered. I obtained the prices by personally asking each Team Manager inside the Operations department.

The formula used to calculate amortization is:

Amortization (e) = Resource price × (1 / Amortization years) × (1 / Project duration (547 h)) × hours used

Item	Price (€)	Amortization years	Time used (h)	Amortization (€)
Dell latitude 7490	1500	5	297	162.88
Dell monitor	190	5	297	20.63
Logitech keyboard	15	5	297	1.62
Logitech mouse	7	5	297	0.76
Office desk	240	20	297	6.51
Office chair	200	20	297	5.42
Chest of drawers	110	20	297	2.98
MareNostrum4	34 million	5	317	1140.26*
At home chair (Ikea)	49	20	198	0.88
At home desk (Ikea)	119	20	198	2.15
Total				1344.09

Table 4
*Table with amortization costs. *MareNostrum4's amortization has been divided by 3456, the total number of nodes, because I am only using a node or even less during the development stages. Source: Own calculations*

For office furniture, I applied a 20 years amortization, as suggested in an article from El País [27]. For the rest of the hardware material, I fixed it to 5 years, because this is the amount of years we have a warranty and/or support with the providers. The time used is based in tasks duration among the project where this material is used.

Electricity cost

I utilize the same laptop plus screen in both of my office and at-home setups. Reviews [28] reported approximately 40W[29] at maximum power consumption. And according to Energy Star [30], the monitor consumes 10.9W when powered on.

Assuming the same price (0.43473 €/kWh [31] at this moment) for both sites. We can estimate my own electricity cost:

*Own electric cost = 0.43473 €/kWh * 4.5 hours a day * (40W + 10.9W)/1000 kW conversion * 110 days*

This makes 10.95€ of electricity cost for my equipment.

[27] https://cincodias.elpais.com/
[28] https://www.notebookcheck.net/Dell-Latitude-7490-i5-8350U-FHD-Laptop-Review.309000.0.htmlDell webpage
[29] I estimate a bit more because my laptop has a more modern processor. I could not find any official data on power consumption
[30] https://www.energystar.gov/productfinder/product/certified-displays/details/2345453
[31] https://tarifaluzhora.es/

Furthermore, the project is developed for MareNostrum4, which consumes a frightening amount of 1.2 million euros worth of electricity a year. [32] Calculating the machine's consumption per hour is extremely difficult because of the difference in power usage among executions. Even though, we can approximate it since my project, except for testing parts, only uses one CPU for development stages. For the rest of MareNostrum's usage, we will assume a whole node.

We know that each CPU consumes 150W [33] when powered. Then we calculate:

*Watts per hour consumed = (1 CPU usage hours * 150W) + (Full node usage hours * 350W) 49050 W/h = (152h * 150W) + (75h* 350W)*

That makes 49.05 kW/h, assuming the same kW/h price as before, the electrical cost for MareNostrum4 reaches up to 21€. Making a total of 31.95€ in this category.

Internet cost

My proportional part of the internet bill at home is 15€. A month contains 420 hours. Considering 4.5h of work in the project a day, for the days worked at home, we can conclude:

*Internet cost = (15€ monthly bill / 420 hours in a month) * 4.5 daily hours of work * 66 days of work at home*

In total, it takes 10.60€ worth of internet during the whole project, when working from home. I do not consider the BSC's internet cost because the company does not pay for the internet, it is supported inside the RedIRIS [34] network.

Work space

I pay 330€ monthly for my flat rental in my village (already divided the whole flat rent by the roomates). I use the equivalent of 1/3 of the space as a workspace (it is teeny tiny). This makes 110 euros a month for my at-home workspace. During the scope of the project, it will entail 550 euros. I do not consider the BSC's building rental because my office placement is already a property of the company.

2.6.3. Other costs

Contingencies

Considering a fund of contingency is a good practice in the case of unforeseen incidences that might take place and a must for most projects. This will be calculated by adding an extra 15% of the contingency margin to our total costs. We have our Generic Cost, 2241.79€, plus 10,285.14€ of the tasks cost. This generates a contingency fund of 15% of (2,241.79 + 10,285.14) = 1879.03€.

[32] All of this information has been extracted from the Operations department
[33] https://www.cpu-world.com/CPUs/Xeon/Intel-Xeon%208160.html
[34] https://www.rediris.es/

Incidental costs

As explained in previous sections, risks and obstacles might appear. But they are already contemplating adding hours to some tasks, to compensate for possible delays or my inexperience with some technologies, as explained in the Risks 2.7 section.

2.6.4. Total cost

All the previously explained costs are joined in the next table, concluding with the total cost of the project. Adding the general costs, calculated with the formula:

Activity	Cost (€)
Cost per Activity	10,285.14
Amortization	1344.09
Electric cost	31.95
Internet cost	10.60
Work space	550
Contingency	1879.03
Total Generic Cost	14103.85

Table 5
Total cost of the project. Source: Own calculations

2.7. Risk management: alternative plans and obstacles

The normal progression of the project might be affected by obstacles. The risks were introduced in the last deliverable, in this one I will explain how these can be surpassed. A level of risk is added for having an idea of how critical these are.

- **High workloads [High risk].** Delays would be easy to foresee since every X weeks, decided by my team leader, I am the supporter on duty in charge to answer all the tickets sent by the BSC users in help. Also, I have other Support tasks to develop that may be urgent. The only solution to this is to make extra hours on my own outside my workday (maybe even in weekends) depending on the week.
- **Inexperience with some technologies and systems [Mid risk].** Most of the systems and technologies used in this project are unfamiliar to me. This requires a lot of time to read the documentation and inspect the code. This is easily solved by contemplating all this time as project tasks (TT1,TT2,TT3, and intrinsically in US7). It is important to have in mind that it will create dependencies between other tasks.

- **Testing and debugging** [**Mid risk**] The testing for seeking bugs and the correction of them actually consumes more time than the main implementation of the code. This is, as before, contemplated in the project's tasks. I estimated each testing task's (US8,US9 and TT6) duration twice as much as I think it would take to me.
- **Good documentation** [**Mid risk**] Creating documentation for this project can be a bit overwhelming since many different documentations must be redacted for different publics. Again, this is contemplated when specifying the project's tasks by adding 2 extra hours to each project documentation task (except PD5 and PD6) to what I consider I would normally need for them .

2.8. Management control

Obstacles and unforeseen events are actually an "expected" part of a project. Because of this, after finishing a task, we will compute the deviation of all the involved costs. For each section, the following formulas have been proposed for each category.

- **Human resources deviation:** Not all days or hours, we as humans perform the same. This formula estimates the deviation in human resources efficiency for each task.

  *Human resources deviation = (Estimated cost per hour - Real cost per hour) * Total hours consumed*

- **Amortization deviation:** Each item can be used more or less depending on the task. Also, its value can vary depending on external economic factors.

  *Amortization Deviation = (Estimated usage hours - Real usage hours) * Price per hour*

- **Electricity cost deviation:** We know, this year more than ever in Spain, that electricity price changes even during the same day. Thankfully, this does not apply to water. Electricity deviation is calculated with the following formula.

  *Electricity cost deviation = (Estimated usage hours - Real usage hours) * Price per hour*

- **Transport deviation:** The same reasoning from the previous bullet point but with gas prices this year. This should be calculated every day when a journey takes place, using the gas price for that day in particular.

  *Transport deviation = (Estimated gas price - Real gas price) * Gas consumption in liters*

- **Total cost deviation:** Groups all deviations in the different tasks. It does not take into account contingencies nor incidents.

  Total costs deviation = Human resources deviation + Amortization Deviation + Electricity cost deviation + Transport cost deviation

With these indicators, we will know where the deviations in the budget are. To get an overview of the project budget we can compare if the total cost of the project is greater than the total estimated cost and see if the contingency margin was estimated correctly.

2.9. Sustainability

2.9.1. Self-assessment

The social and environmental dimensions of ICT always have been a concern of mine. The huge pollution of electronic waste and the immense and profound impact of technology in our lives are problems rooted in our society that seem so far away to be solved.

By doing the self-assessment survey provided by the UPC, I noticed that I actually might not know as much about these aspects as I thought, plus I am barely aware of the economic dimension of it. First of all, doing the budget for this project, there are so many factors that I did not even bear in mind at first, like my in-home expenses from when I work from home, which is as valid as the ”real” office expenses, of which I shall appreciate some expenses that I normally I do not consider (I.e the desk where I sit). In summary, the fact that I do not personally pay for some of the equipment that I daily use obviously has a cost that must be taken into account. Also, I became interested in all the different economic indicators of a project and learned to understand and to develop an economic viability report, so I can do better with my future projects.

For the environmental dimension, even though I am conscious of the huge repercussions of technology in our world, I realized that I do not know as much as I would about reducing pollution and electric consumption apart from the basic advice such as taking your electronic garbage to a green point and make better codes.

2.9.2. Economical dimension

Regarding PPP: Reflection on the cost you have estimated for the completion of the project: The reflection of the cost I estimated can be found in the Budget 2.6 section. It considers the human and material costs, as well as other costs that do not fit into these two categories. The salaries and the cost of the material have been asked to the personnel involved. The amortization and the other costs are an estimation since there is not a 100% accurate way to calculate them. I tried to make the budget as realistic as I could to have a real reference of what this project implies.

Regarding Useful Life: How are currently solved economic issues (costs...) related to the problem that you want to address (state-of-the-art)? The funding for this project comes exclusively from the BSC, so if any issue occurs or some extra resource is needed (time included), it is possible to talk to Project Managers and see if it is possible to achieve that required resource. This type of cost falls within the field of unforeseen costs.

Regarding Useful Life: How will your solution improve economic issues (costs ...) with respect to other existing solutions? Other performance analysis tools are developed by many large developer Teams. This whole project is conceived for being developed by only one person, this implies that is cheaper and has the optimization and light overhead design as a starting point, which translates into much less costs for energy and resources.

2.9.3. Environmental dimension

Regarding PPP: Have you estimated the project's environmental impact? We have an idea of the resources that are needed to carry on my project, but we cannot estimate with total accuracy and security how it will affect the environment, there are factors that we have no way to estimate, or the impact might be deeper than we expected. P.e, we know how much electricity MareNostrum4 consumes, but we cannot know exactly the electricity "greenness" and the environmental impact that causes in its generation. What we can assure is that if execution times are reduced, more hours could be given to other projects that will be allocated in MareNostrum4.

Regarding PPP: Did you plan to minimize its impact, for example, by reusing resources? The whole project has been conceived to follow the best practices for performance, so that not that much electricity is wasted. This also applies to my personal laptop and work equipment, trying to save as much energy as possible by shutting down electronic devices when not needed or decreasing my screen's brightness. Other physical resources for this project are not used.

Regarding Useful Life: How is currently solved the problem that you want to address (state of the art)? How will your solution improve the environment with respect to other existing solutions? Our project will help programmers with identifying issues within their codes so they can improve their performance. Thus, their codes will become more efficient and more energy is saved.

2.9.4. Social dimension

Regarding PPP: What do you think you will achieve -in terms of personal growth- from doing this project? I personally love from my day-to-day job the idea of helping researchers to do their science and being part in some way of new discoveries. With this project, this aspect will acquire a new dimension by helping researchers to make better codes. I will learn technologies that are not familiar to me and reinforce my skills with the ones I already know. I am grateful for all of these parts of the project and I feel more mature as an engineer for every step that I take forward.

Regarding Useful Life: How is currently solved the problem that you want to address (state of the art)? How will your solution improve the quality of life (social dimension) with respect to other existing solutions? I hope that my project removes a bit of conscience inside researchers to start being more conscious of their codes and the impact of resource waste. Other performance analysis tools might exist, but this integration and expansion are very particular inside the BSC ecosystem, where I think the most impact will generate.

Regarding Useful Life: Is there a real need for the project? I do believe in the necessity of this project. TALP is a reference and a pioneer for lightweight analysis tools. No other analysis tools can be really compared because of the usability and the barely introduced overhead, and contributing to it and making it more accessible will surely help people concerned about their codes. Nonetheless, resource waste is a big problem in this digital era that must be solved, or at least, reduced as much as we can.

3. UserPortal's integration

In this section, I will explain the development of the User Support part.

3.1. Actual system design

UserPortal's actual back end involves a set of projects, services and systems that work across machines and nodes inside them to generate, gather and process information. It is essential to consider this as a guide on what can or cannot be done, to not redo mistakes and to improve the service.

There are, in general terms, three machines implied:

- UserPortal's Virtual Machine - Virtual machine only accessible by the Operations Team which contains all the services required for running UserPortal: SQL Server, Apache Server, InfluxDB, daemons and a specific Python environment.
- Cluster's head - Separated node (definition found in Glossary A)from each cluster, accessible only by the Operations Team, which serves for OS management purposes, monitoring services and as an access point to the rest of the nodes. [35]
- Data Storage Machine - Cluster at the BSC intended for data storage and data moving. In this situation, it is used as an access point for the UserPortal's virtual machine because it has no direct access to MareNostrum's head.

The following diagram synthesizes the structure and the execution done by the services:

[35]In the BSC machines, you cannot access a compute node directly from a login node. A request for an allocation is needed first.

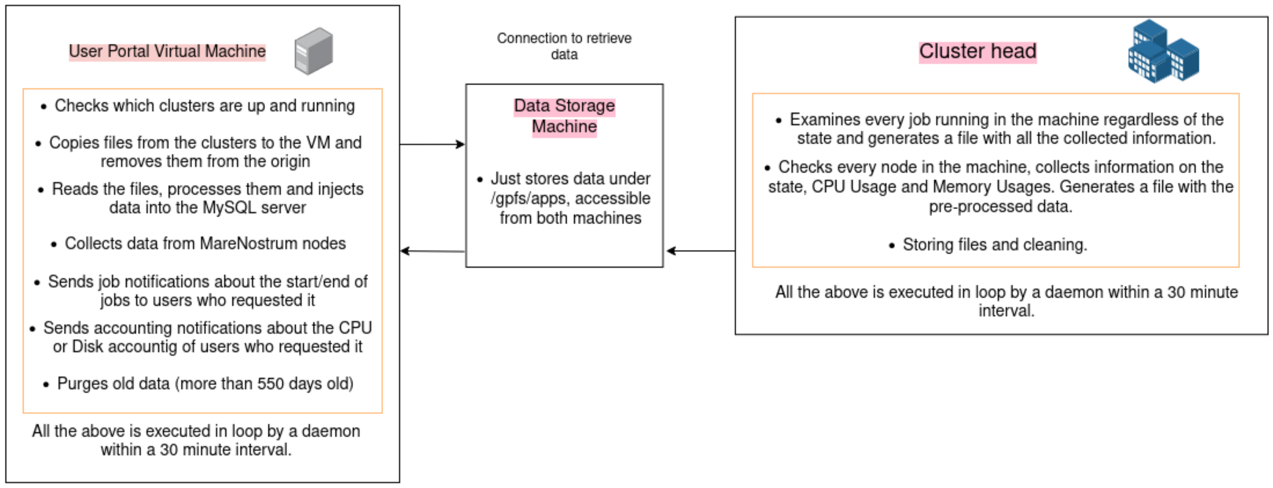

User Portal Virtual Machine
Checks which clusters are up and running
Copies files from the clusters to the VM and removes them from the origin
Reads the files, processes them and injects data into the MySQL server
Collects data from MareNostrum nodes
Sends job notifications about the start/end of jobs to users who requested it
Sends accounting notifications about the CPU or Disk accountig of users who requested it
Purges old data (more than 550 days old)
All the above is executed in loop by a daemon within a 30 minute interval.
Connection to retrieve data
Data Storage Machine
Just stores data under /gpfs/apps, accessible from both machines
Cluster head
Examines every job running in the machine regardless of the state and generates a file with all the collected information.
Checks every node in the machine, collects information on the state, CPU Usage and Memory Usages. Generates a file with the pre-processed data.
Storing files and cleaning.
All the above is executed in loop by a daemon within a 30 minute interval.

Figure 6
Diagram summing up each machine service execution. Source: Own

3.2. Design decisions

The original idea was to integrate the whole system into the existing back end, but some issues influenced the decision of making it a separate system.

- The actual backend is written in Python 2.7[36] (plus old dependencies) - As developers, we have the responsibility of not perpetuating the use of deprecated software as it may lead to malfunction of the system or make it vulnerable. The Python part of the actual project will be written in Python 3 and if someday the backend gets ported to Python 3 I will consider to integrate it.
- The backend is very complex and designed for running every 30 seconds with a heavy payload of data to process. What we need is more lightweight and simple.
- File ownership - TALP generated files are owned by the users in compute nodes once the execution of their programs finishes, they are generated by the backend. UserPortal generates all the information by calling the SLURM API, and stores it in files in the shared filesystem in a field where only Operations has writing permissions. This is a huge change of approach. In the wrapper (3.4.1) subsection, this problem is explained in detail.
- Portable and independent - The actual deployment is manual and tedious, also sysadmin-dependant. With this implementation, only a few changes have to be made to port it across machines.

[36]This version of Python has been officially deprecated since 2020 https://www.python.org/doc/sunset-python-2/

3.3. Requirements

The key concepts that we want to take as a requirement during the whole development can be summed up in the following table:

Requirement	Definition in this context	Decisions taken to achieve it
Portable	- Easy to install/deploy across machines - Easy to adapt for each machine	- Using Python and Bash as languages, all machines have them installed - General scripting - only a few environment variables change across machines - CI/CD for automated and easy deploy
Secure	- Error-proof - Will not overflood the system (memory and performance) - Will not break the system (malfunction, vulnerabilities...) - Will not corrupt database	- Error control at all levels and secure coding - Junk collector and proper cleaning of files - Efficient programming - Exhaustive testing before production deploy
Scalable	- An unexpected increase in the files to process won't collapse the system and performance won't be degraded "that much"	- Efficient coding and technology - KISS philosophy

Table 6
Summary of the key concepts in the design

3.4. Backend development

With all these concepts in mind, the actual development can begin. To summarize, a system is needed that makes it transparent for the users to load TALP and view the results on their UserPortal job page once their job has completed

TALP generates different reports based on user's needs. Users can choose between POP metrics, a summary at the node level, summary per region, raw data or any combination of them. The report of interest for this project is the pop-metrics report, which provides a summary of the useful computation time of the application, time spent in MPI communication, and load imbalance. These are the metrics that we want to focus on and educate developers to understand. Now, following the schema of the actual UserPortal backend. We will need, at a first glance:

- A wrapper and module for the command mpirun to pre-load the necessary libraries and configure TALP options.
- A script to gather the data and pre-process it.
- A script to parse the TALP file and inject the data into the DB.
- A cron to execute the services in UserPortal automatically and periodically.

The reason why I decided to separate the gathering of files and the processing of data is that each of them is written in a different programming language.

3.4.1. Wrapper

The main objective of this wrapper is to make the TALP configuration as transparent as possible for MareNostrum 4 users. TALP works with MPI, so we will need to make a wrapper for running an application with MPI. This command is "mpirun", and it is available in all the installed MPI software in the BSC's clusters, both the OpenSource implementation (OpenMPI) and the proprietary implementation from Intel (IntelMPI). We will also implement a wrapper for "srun", the SLURM command for executing jobs.

The wrapper needs to, apart from configuring the arguments and the libraries, generate and place the generated file to a place where we can retrieve it easily. Compute nodes are very limited in terms of connection, they have no access to internet and they can only connect with MareNostrum4 login nodes.

This creates a conflict because TALP-generated files are owned by the user who executes the job, so we need a directory where everyone can write, the disk space won't run out easily and we can clean once the files are gathered. The first thought was to copy the files to some location in GPFS (definition can be found in the Glossary A), like UserPortal does inside the /gpfs/apps partition. This would made the transfer between compute nodes and login nodes much more easy, but we do not want anyone to be able to write inside the Operations-protected area nor any particular group's /gpfs/projects or /gpfs/scratch files, which store important and voluminous data, apart from being restricted with a disk quota per user. This paths will be visible from the wrapper so it is not a secure option. Additionally, GPFS is a very sensitive to high workloads of read/write of files in a same location, as we cannot know how many people will be using the wrapper it is better to store the files in another place.

The best solution that we found is to store files in the /scratch/tmp partition of each login node. This partition is already set with the *sitcky bit*[37] so anyone can write inside it, but when we need to do the cleanup, we won't be able to delete the files. So, what we did is to create a hidden directory inside /scratch/tmp called ".talp", owned by usertest [38]. It is not a problem if somebody inserts junk into the folder because we will only retrieve files following a certain name pattern, formed by the job ID and the machine where it has been generated, the rest will be cleared up. The gathering script controls if the folder is deleted, and creates a new one with the right permissions if it is necessary. This situation might happen as well after a node reboot.

The process of moving the file from the compute nodes' storage to the new directory is summed up in the following diagram:

[37]This allows any user to write in the directory and own the exclusive permissions for that file, so only the owner can edit it, rename it or delete it.

[38]User Support machine user that performs automated tasks and tests, we do not have root

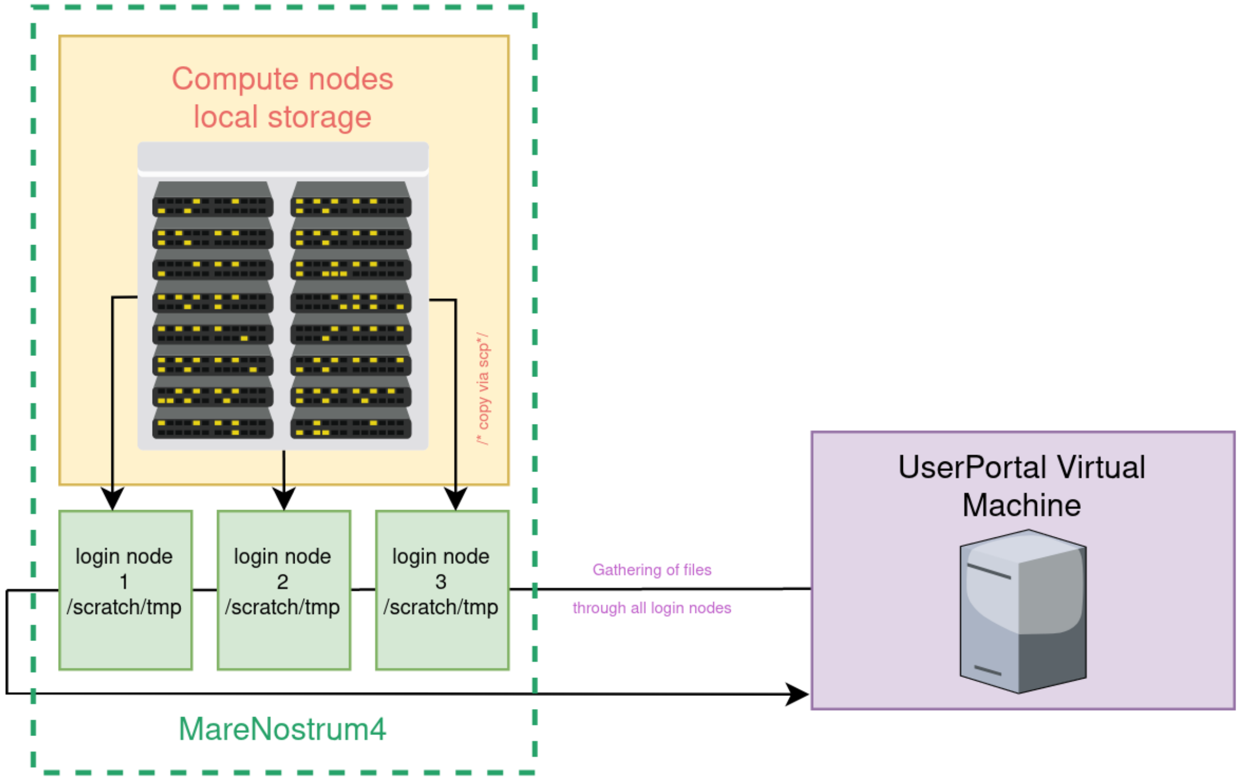

Compute nodes local storage
/* copy via scp*/
login node 1 /scratch/tmp
login node 2 /scratch/tmp
login node 3 /scratch/tmp
Gathering of files
through all login nodes
UserPortal Virtual Machine
MareNostrum4

Figure 7
Location and moving pipeline of TALP-generated files. Source: own

Apart from these core functionalities, the wrapper detects which MPI executable has to be executed by determining the loaded implementation by checking the "MPI" environment variable. Which can only take the values "impi" or "openmpi". This environment variable is set inside the module from each software for all the versions we have installed in MareNostrum. Also if we want to activate PAPI counters.

So, once the execution of a program finishes its execution and the TALP report is generated, it copies[39] the file from the compute node storage to the /scratch/tmp partition of one of the login nodes of MareNostrum. We have three login nodes, and the decision of going to one or another is taken by performing the modulo operation with the job ID. This way the storage is fairly shared among the nodes. It might look a bit scandalous this decision because afterwards there is then the triple of work in performing connections at the gathering phase, but we considered this the best option because even if we loose efficiency, it is better to loose more time in connecting to the three nodes than risking to overload one login filesystem and provoke machine malfunctioning.

Once everything is set, the program is run and the TALP-generated file follows the explained pipeline until the data is inserted in the UserPortal's database.

[39]It is secure copied connecting to the node because compute nodes and login nodes do not share any filesystem.

3.4.2. Gathering Script

The process of gathering data will utilize system commands for ease and efficiency, using the bash scripting language. A script will be created to retrieve TALP-generated files from MareNostrum 4 to the UserPortal's virtual machine, and a logfile A will be kept to facilitate the detection of any malfunctioning or unexpected behaviour.

Firstly, it is important to check if the directory located inside each login node's /scratch/tmp directory exists. If it does not exist, it will be created with the necessary permissions, as previously explained. The core command used for data retrieval is "rsync", which will perform the copy from the MareNostrum folder to a local folder on the virtual machine for temporary storage of gathered files. "rsync" was chosen over "scp", which was found to be faster, due to its "–remove-source-files" option, which automatically deletes the file after it has been copied from the source machine. Additionally, the output of the command can be stored and formatted for logging, which is not possible with the "scp" command, as it does not provide a summary of transferred files.

For the logging aspect, files will be organized on a weekly basis. If the number of users utilizing the tool increases, a daily log will be considered. Within each log, it will report every time the script is run, detailing which files were transferred and from which login node. If no files were transferred, this will be logged as well. Additionally, if the creation of a folder was necessary, it will be noted in the log.

3.4.3. Parsing-Injecting Script

For parsing and injecting the data, the idea is to work with Python because it offers many different libraries to work with SQL connection and parsing files. In bash there is no easy way to do that.

There are 4 different formats to choose from in TALP to print the output: JSON, XML, CSV or plain text. The CSV format creates a different file for each report, difficults the previous gathering and parsing. The plain text format and XML are difficult to parse in Python. So JSON is the best option of format.

After the format was decided, it was relatively easy to implement the script because Python offers a package to work with JSON files. The script itself it is easy in concept, it opens every file in a folder whose name matches the known pattern, and parses the JSON fields getting only the parameters we want. Then those are inserted into the database.

The important thing here was to add security systems to ensure that the database will not be compromised. This englobes ensuring that data is correct and free of errors, checking the values of variables before they are used, testing software in edge or corner cases, a try-catch structure to handle errors in a controlled way, implementing debugging modes to help identify and fix bugs and security vulnerabilities, also with specific error messages to make it easier to detect bugs and issues. Also, command timeouts in case the connection with the database hangs, this way we prevent stuck processes left hanged the machine.

When all procedures have been done, we can be sure about injecting valid data into the database. After all of it, parsed files are deleted.

3.4.4. Module and cron

For MareNostrum 4, we need a module to make the necessary exports to include the location with the wrappers in the "PATH" environment variable. This will be the only environment variable needed because the rest of the configuration is set up inside the wrapper.

For the cron, I created a crontab that will run every 5 minutes in UserPortal's VM. Before executing both the gathering and the parsing-injecting script, it sources the environment to use a Python installation in version 3.11[40] with the packages I require.

3.5. Database Adaptation

UserPortal retrieves data from two main Databases. A MySQL (A) one and an InfluxDB (A) one. The SQL database stores general data such as the jobs and nodes information, past maintenances, accounts and machines, etc. InfluxDB stores voluminous and specific data from jobs as all the points to plot the CPU and Memory usage, so we are not interested in touching this DB at the moment.

If in the future we decide to store TALP bigger amounts of data like the metrics per node, we will revisit it in the future.

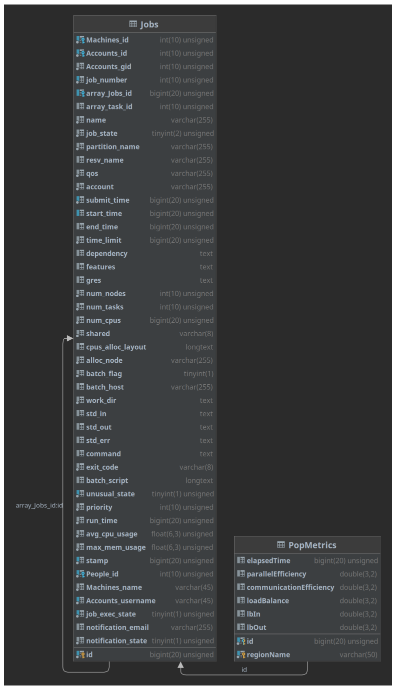
It is important to keep in mind that a user can define various monitoring regions [41] for TALP, because this influences the design of the table.

With this information, for collecting POP Metrics [42], the best option was to create a new table inside the SQL database. We decided to be a separate table mainly because of the regions we mentioned before. If we add the TALP data to the "jobs" table (which already contains a lot of fields), we will be limited to only 1 Monitoring Region per job. By creating a Primary Key pair composed of the UserPortal's job ID and the Monitoring Region name, the user can define as many regions as it may need. The schema of the two implied tables in the database is:

[40]This version is chosen because its the latest stable release which features a huge efficiency improvement in comparison with previous versions

[41]Region inside the code whose performance will be traced. The default monitoring region is the MPI monitoring region, which goes from MPI_Init to MPI_Finalize. The users can define their own monitoring region through the use of the TALP API

[42]If you want to learn more about them you can visit https://pop-coe.eu/node/69



Jobs	
Machines_id	int(10) unsigned
Accounts_id	int(10) unsigned
Accounts_gid	int(10) unsigned
job_number	int(10) unsigned
array_Jobs_id	bigint(20) unsigned
array_task_id	int(10) unsigned
name	varchar(255)
job_state	tinyint(2) unsigned
partition_name	varchar(255)
resv_name	varchar(255)
qos	varchar(255)
account	varchar(255)
submit_time	bigint(20) unsigned
start_time	bigint(20) unsigned
end_time	bigint(20) unsigned
time_limit	bigint(20) unsigned
dependency	text
features	text
gres	text
num_nodes	int(10) unsigned
num_tasks	int(10) unsigned
num_cpus	bigint(20) unsigned
shared	varchar(8)
cpus_alloc_layout	longtext
alloc_node	varchar(255)
batch_flag	tinyint(1)
batch_host	varchar(255)
work_dir	text
std_in	text
std_out	text
std_err	text
command	text
exit_code	varchar(8)
batch_script	longtext
unusual_state	tinyint(1) unsigned
priority	int(10) unsigned
run_time	bigint(20) unsigned
avg_cpu_usage	float(6,3) unsigned
max_mem_usage	float(6,3) unsigned
stamp	bigint(20) unsigned
People_id	int(10) unsigned
Machines_name	varchar(45)
Accounts_username	varchar(45)
job_exec_state	tinyint(1) unsigned
notification_email	varchar(255)
notification_state	tinyint(1) unsigned
id	bigint(20) unsigned

PopMetrics	
elapsedTime	bigint(20) unsigned
parallelEfficiency	double(3,2)
communicationEfficiency	double(3,2)
loadBalance	double(3,2)
lbIn	double(3,2)
lbOut	double(3,2)
id	bigint(20) unsigned
regionName	varchar(50)

Figure 8
Jobs and PopMetrics table in the UserPortal database. Source: DataGrip-generated

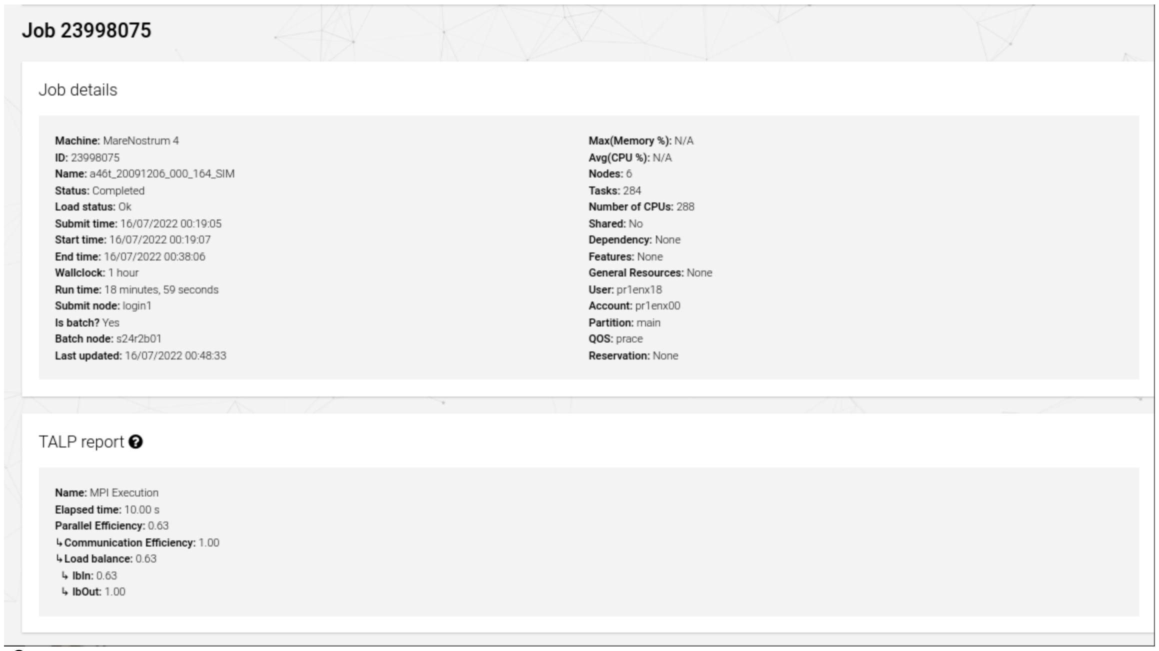
3.6. Web development

Understanding UserPortal's front end has been one of the most arduous tasks for me since I had no expertise in web development or PHP.

The general structure follows the classic Data - Domain - Presentation layer design. A particularity of this webpage is that it does not use any framework and everything is coded barehandedly. Each layer contains at the same time ”microlayers” with controllers, templates and services.

After a lot of time understanding how each layer worked and studying about everything that should be adapted, changes were made to the internal schematic of the Domain Layer so we could have a new PopMetrics class. Its main methods consist of searching by the job's ID since we do not need to operate with the data.

Finally, inside the job's view, a new container with TALP data is only shown if it exists at least one entry in the PopMetrics table with data for that job. It prints the monitoring regions all at once.



Job 23998075

Job details

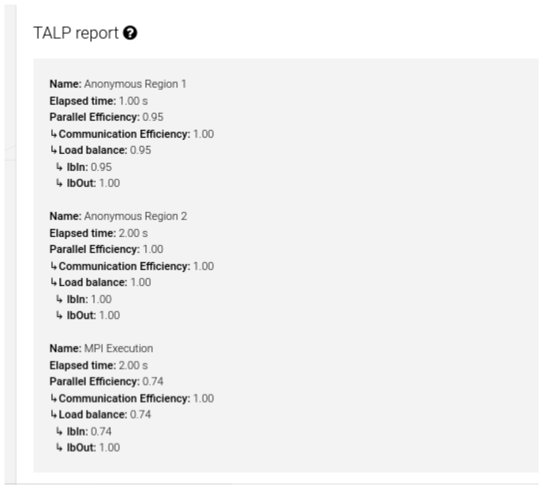
Machine: MareNostrum 4
ID: 23998075
Name: a46t_20091206_000_164_SIM
Status: Completed
Load status: Ok
Submit time: 16/07/2022 00:19:05
Start time: 16/07/2022 00:19:07
End time: 16/07/2022 00:38:06
Wallclock: 1 hour
Run time: 18 minutes, 59 seconds
Submit node: login1
Is batch? Yes
Batch node: s24r2b01
Last updated: 16/07/2022 00:48:33

Max(Memory %): N/A
Avg(CPU %): N/A
Nodes: 6
Tasks: 284
Number of CPUs: 288
Shared: No
Dependency: None
Features: None
General Resources: None
User: pr1enx18
Account: pr1enx00
Partition: main
QOS: prace
Reservation: None

TALP report

Name: MPI Execution
Elapsed time: 10.00 s
Parallel Efficiency: 0.63
↳**Communication Efficiency:** 1.00
↳**Load balance:** 0.63
↳ **lbIn:** 0.63
↳ **lbOut:** 1.00

Figure 9
Screenshot of a job webpage where it shows the TALP report with the POP Metrics. Source: Own



TALP report

Name: Anonymous Region 1
Elapsed time: 1.00 s
Parallel Efficiency: 0.95
↳**Communication Efficiency:** 1.00
↳**Load balance:** 0.95
↳ **lbIn:** 0.95
↳ **lbOut:** 1.00

Name: Anonymous Region 2
Elapsed time: 2.00 s
Parallel Efficiency: 1.00
↳**Communication Efficiency:** 1.00
↳**Load balance:** 1.00
↳ **lbIn:** 1.00
↳ **lbOut:** 1.00

Name: MPI Execution
Elapsed time: 2.00 s
Parallel Efficiency: 0.74
↳**Communication Efficiency:** 1.00
↳**Load balance:** 0.74
↳ **lbIn:** 0.74
↳ **lbOut:** 1.00

Figure 10
Screenshot of a job webpage where it shows the TALP report with the POP Metrics and more than one monitoring region. Source: own

3.7. CI/CD

A concept we had in mind for this part of the project is to make deploys as easy as possible. The scripts go to different machines, so we would have to pull manually the repository in at least 3 different locations (UserPortal VM, Marenostrum 4 TALP module directory and DLB module inside MareNostrum 4).

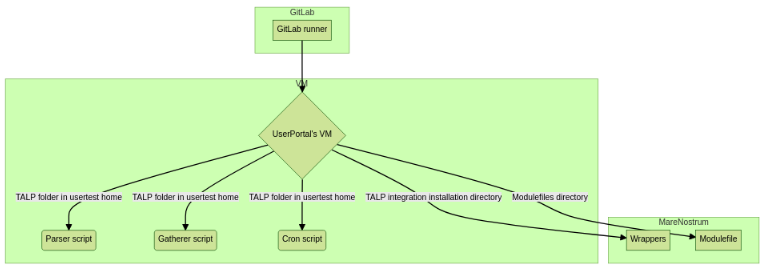


Figure 11
Visual representation of connections made by GitLab runners.
Source: own

The best option is to set up GitLab CI/CD, and work always in a local location, not in any production environment directly, this will make development and deployment more secure. The workflow of CI/CD is: everytime there is a push in the repository, to have a deploy stage for each file, which would be copied to its final location. We do not allow gitlab-runners to run directly into HPC-Machines, so we need to use an intermediate trusted machine that will copy the files from inside them to the final cluster location. UserPortal's VM serves this purposes and also simplifies the number of connections to be made.

3.8. Testing

As explained, exhaustive testing is the minimum required to assure that we are not compromising the system or database's integrity. The main issue with the database is that the test database is updated with new jobs every day at 12 AM, so this delays the testing with recently executed jobs.

With the individual components, I made sure that all methods to make the scripts secure were doing their job and that all bugs were corrected. I made a test case to find any corner case:

- Run the script and check that the data is correctly inserted into the test database. Verify that the data inserted is the same as the data in the test file.
- Create a test file with data that is not in the correct format and run the script. Verify that the script exits gracefully and does not insert any data into the database.
- Create a test file with data that is in the correct format, but contains unexpected fields. Run the script and check that it only inserts the fields that are expected and does not throw any errors.

- Create a test file with data that is in the correct format, but contains invalid data. Run the script and check that it exits gracefully and does not insert any data into the database.
- Create a test file with data that is in the correct format and run the script, check the script exits gracefully and does not insert any data into the database if the job id is not found in the database.
- Create a test file with a filename which is not in the correct format and run the script, check the script exits gracefully and does not open nor insert any data into the database.

For the whole system, I not only tested it myself but I asked my colleagues to make a few experiments. Every bug reported was corrected and I also took suggestions, after guaranteeing that everything worked fine I proceeded to put everything in production.

4. TALP - PAPI extension

4.1. Understanding how TALP works

In this part of the project, the most arduous part with a difference has been the understanding of the entangled code of TALP. The main code is found in a single file [43] and it is difficult to trace the execution. We also need to know about what structures exist, in order to modify them properly, here is a diagram I made for easy understanding.

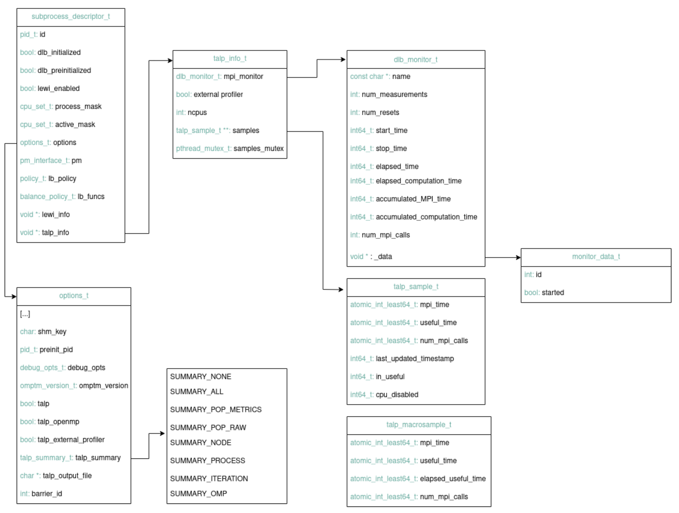


Figure 12
DLB-TALP own data structures.
Source: own

As we can see, TALP creates a structure called talp_info_t, which encompasses a structure for the moni-

[43] Except for the output printing, found completely in another file of the library

toring region (dlb_monitor_t) and a pointer to one or more structures called samples. Each thread creates a sample and it is owned and updated exclusively by that thread. The structure talp_macrosample_t contains the accumulation of metrics from all samples within that Monitoring Region. The structure dlb_monitor_t contains the metrics from the Monitoring Region, calculated at certain moments of the execution with the information inside their respective macrosamples.

The workflow for this project is highly complex and involves multiple data structures and processes. To gain a better understanding, I created a detailed flowchart for myself to visualize the various steps and components involved. This can be found in the Appendix (19). To further simplify this, I also created a condensed summary flowchart that provides a succinct overview of the entire workflow.

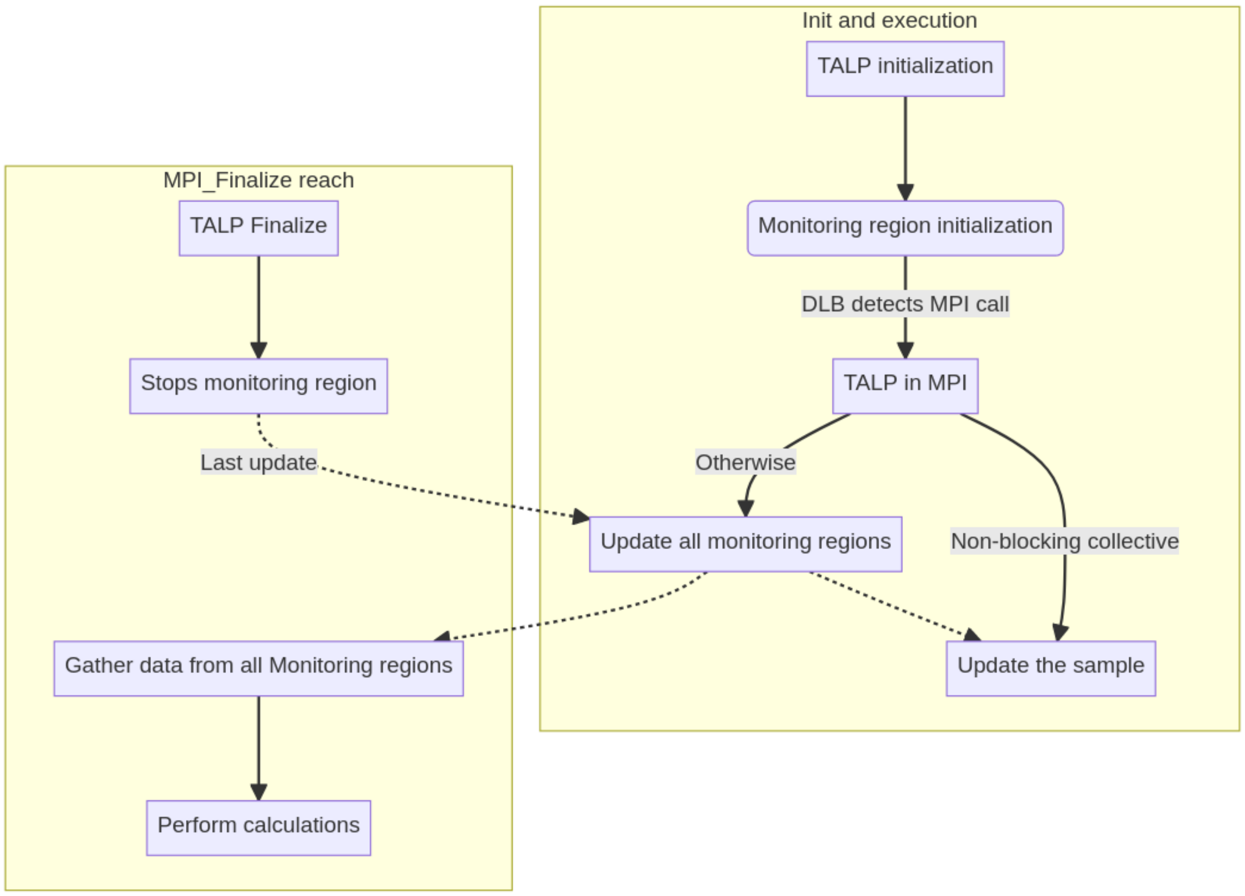


Figure 13
TALP workflow simplified chart.
Source: own

1. The library and monitoring regions are initialized.
2. MPI Monitoring region is initialized and started once we run into MPI_Init.
3. We run into MPI calls:
   (a) If the call is a non-blocking collective, we just modify the thread's sample, by updating the

calculation of the useful time / MPI time.

(b) Otherwise, it updates all the monitoring regions. Updates the sample, accumulates all samples into a macrosample and updates the MPI monitor.

4. We reach into MPI_Finalize
5. A final gathering and update of all data/samples is performed
6. Final calculations for the metrics are made
7. Prints the output

4.2. Autoconf

We do not have to forget that in order to implement PAPI into TALP, first TALP needs to include the PAPI library and headers. TALP is configured using Autoconf [44], an Open Source project to automatically configure software source code packages.

The structure is a bit complex requiring a more in-depth study than initially anticipated. Adjustments had to be made to allow TALP to accept the "–with-papi" option during configuration. Additionaly, I defined the necessary macros(definition can be found in the Glossary A) to define region codes that will only be compiled when the aforementioned option is explicitly specified.

4.3. Implementation

Before beginning the implementation process, a thorough study of PAPI was conducted as I had no prior experience with it. Through this research, I gained an understanding of the various ways PAPI could be utilized and the breadth of information it could provide. To further solidify my understanding, I wrote and executed various test programs in C to gain hands-on experience with the library.

PAPI[45] has ben integrated into TALP in the following way:

Firstly, it should be noted that TALP can be used without MPI and users who want to track their application's performance in terms of Instructions Per Cycle should be able to use it as well. Therefore, PAPI is separated from the MPI Monitoring region by default.

Next, it is important to be clear on how we intend to use PAPI. The library offers different levels of interfaces, a high-level API that is easy to program but not customizable, and a low-level API that is more difficult to program but allows features such as EventSet [46] creation and thread-level tracing.

[44]https://www.gnu.org/software/autoconf/
[45]Version 6.0.0 has been used during the whole project
[46]Bit mask used internally by PAPI containing the desired counters we want to use

The main idea is to create an EventSet that will englobe the PAPI_TOT_CYC [47] and PAPI_TOT_INS [48] counters. Additionally, PAPI is activated for thread-level tracing. When an MPI call is encountered and a sample is updated, we stop reading the counters since we are not in useful time. The values from the counters are then read and stored inside the "cycles" and "instructions" fields, which have already been created in the talp_sample_t structure. Once this is done, the counter's value is reset to 0 and counter tracing is restarted at the end of the function.

It is important to remember to implement the accumulation of metrics from the samples in the macrosample structure. To do this, we will add two new fields inside the macrosample's structure for storing the accumulated cycles and instructions, respectively.

Once the execution is finished and TALP is performing the final gathering of data, we will calculate the IPC from the accumulated values of macrosamples, being the formula of IPC:

$$Instructions\_per\_Cycle = \frac{Instructions}{Cycles} \tag{1}$$

This value is stored inside the dlb_monitor_t structure inside a new field called "ipc" and later displayed when output printing.

A significant portion of the implementation process involves the implementation of all possible ways to print the output and other performance metrics of interest such as the raw data, the plain number of cycles and instructions. Each type of report can be printed in different formats, JSON, XML, CSV or plain text, and performs MPI operations for collecting data across nodes or processes, like a reduce [49] operation or a gather [50]one. It even defines unique MPI datatypes It is important to test that this part works correctly because otherwise the traced data can be lost in communication.

It is important to check other functions that can be called by the user using the API as they might require an extra update or reset of the sample. Additionally, it is essential to remember to properly shut down PAPI in order to free up any resources it may be using. In total, five files from the project have been modified in order to properly implement PAPI counters within TALP.

The next diagram synthetizes the PAPI additions to the TALP execution flowchart:

[47]Total cycles
[48]Total instructions
[49]MPI_Reduce is a collective operation in MPI that is used to combine values from different processes into a single value
[50]The MPI_Gather operation is a collective operation that is used to gather data from all processes and combine them into a single buffer on one process.

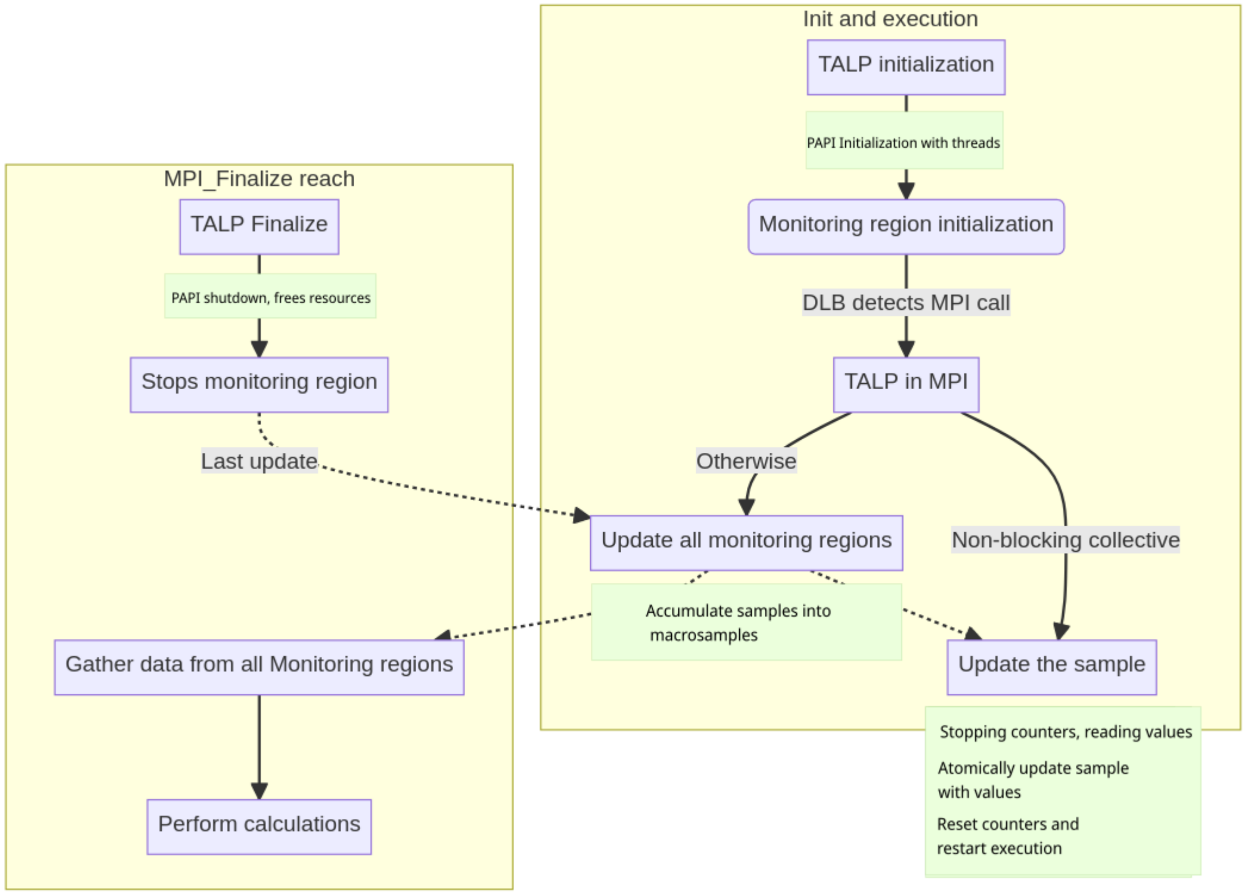

Init and execution
TALP initialization
PAPI Initialization with threads
Monitoring region initialization
DLB detects MPI call
TALP in MPI
Otherwise
Non-blocking collective
Update all monitoring regions
Accumulate samples into macrosamples
Update the sample
Stopping counters, reading values
Atomically update sample with values
Reset counters and restart execution
MPI_Finalize reach
TALP Finalize
PAPI shutdown, frees resources
Stops monitoring region
Last update
Gather data from all Monitoring regions
Perform calculations

Figure 14
TALP workflow simplified chart with PAPI integration.
Source: own

To be precise on whether functions have been altered with the implementation, in the Appendix 20 can be found the big chart with the execution edited. The functions with alterations are marked in purple.

4.4. Testing

The most important aspect of integrating PAPI into TALP is ensuring that the counters are being stopped, read, reset, and started again correctly.

Gradual testing and the use of debugging messages can help to confirm that when an MPI call is encountered, a meaningful value is read and properly stored and accumulated. Then, ensure that the counters in the EventSet are reset to 0, and this should be done inside each thread to avoid interference with counters from other threads. Additionally, it is important to stress the importance of error control to ensure that nothing is overlooked.

4.4.1. Data correctness and validation

While PAPI can be used to trace cycles and instructions, it is not possible to confirm their accuracy independently. However, the in-house tracing software, Extrae, can also be configured to trace PAPI counters and calculate Instructions Per Cycle (IPC) for useful computation regions.

By performing tests and comparing the outputs of Extrae and TALP, we found that the results were similar but with slight deviation (not greater than 0.05 points between executions). It should be noted that Paraver reports an IPC interval rather than a fixed number.

What differs from execution to execution is mainly the cycles number. This teeny tiny difference can be explained by some factors:

- External factors. Processes running in the system at the same time can interfere with our executions and affect the number of cycles used by our program.
- Memory access patterns. The way our program accesses memory can have an impact on the number of cycles. For example, more cache misses imply more time to access memory, and in consequence, more cycles used.
- Non-deterministic execution. Even if the input is the same, some events can cause the program to take different paths during execution like branch mispredictions. This may lead to variations in the number of cycles used.

4.5. Overhead evaluation

It is known that PAPI can introduce overhead to program execution as it uses the syscall read() [51] to read hardware counters. As TALP is a lightweight tool, it is crucial to evaluate the amount of overhead that it introduces in order to determine whether or not it is necessary to notify the user about the increase in execution time. This can be accomplished by measuring the execution time of the program both with and without PAPI enabled, and comparing the results.

It is also important to note that the overhead introduced by PAPI may vary depending on the specific counters used, the number of events in the eventset, as well as the architecture and platform on which the program is running.

To evaluate the overhead introduced by PAPI, we used three benchmarks: PILS (distributed alongside DLB), Linpack, and HPCG. All experimentation was performed in the same manner for all benchmarks. They were compiled and executed with the same Intel stack, version 2018, which has been found by the User Support Team to provide better performance and reliability in MareNostrum4[52]. For each

[51]https://man7.org/linux/man-pages/man2/read.2.html
[52]https://www.bsc.es/supportkc/tech-reports/mpi-report

benchmark, the same exact binary was run on 1, 2, 4, 8, and 16 nodes, starting 48 [53] threads per each MPI task and one MPI task per node.

The results were calculated by performing at least 20 executions of each benchmark and then calculating the median of the interested metric, with the maximum and minimum times removed from the calculation. Graphs were generated with the results, making it easy to compare the three execution scenarios of the benchmark. The plain execution is represented in blue, execution with TALP but without PAPI counters enabled is represented in red, and execution with both TALP and PAPI enabled is represented in yellow.

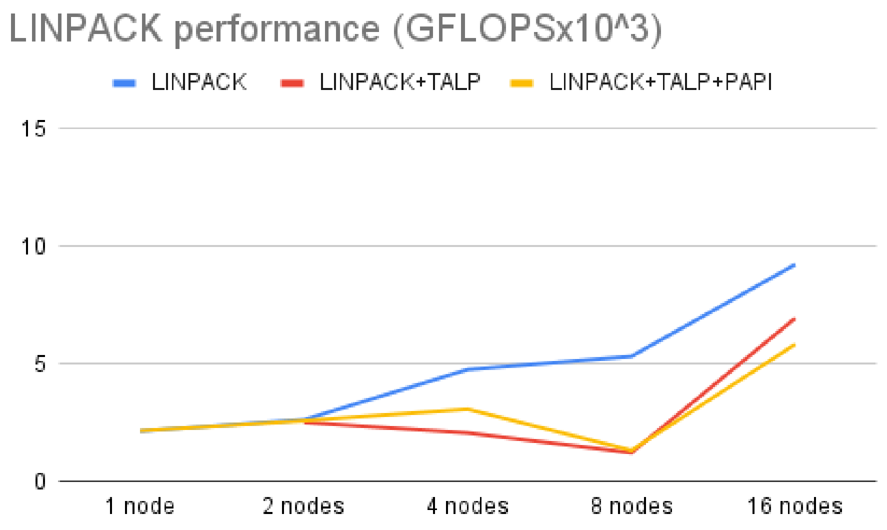


Figure 15
LINPACK performance in GFLOPs in different scenarios and number of nodes used in the execution. Source: own

[53]Note how MareNostrum4 processors have 48 cores per thread

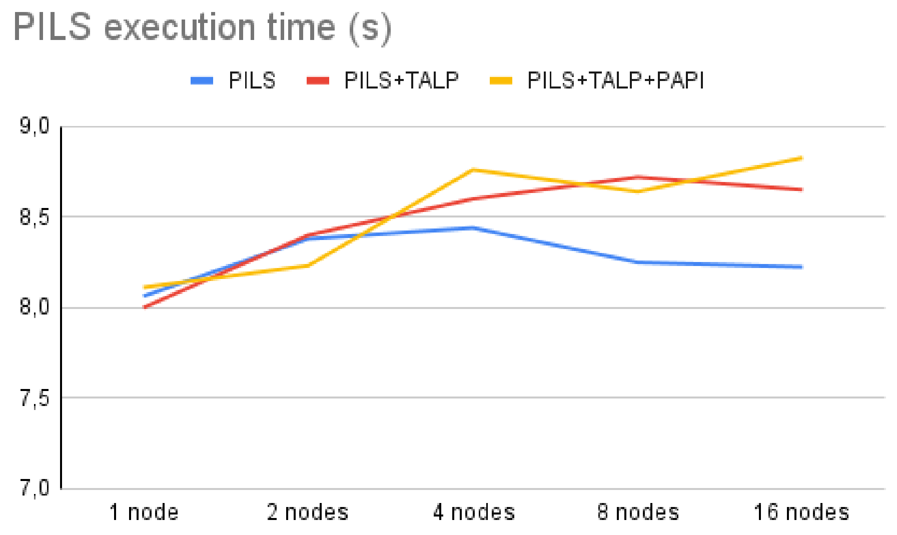


Figure 16
PILS execution time in seconds in different scenarios and number of nodes used in the execution. Source: own

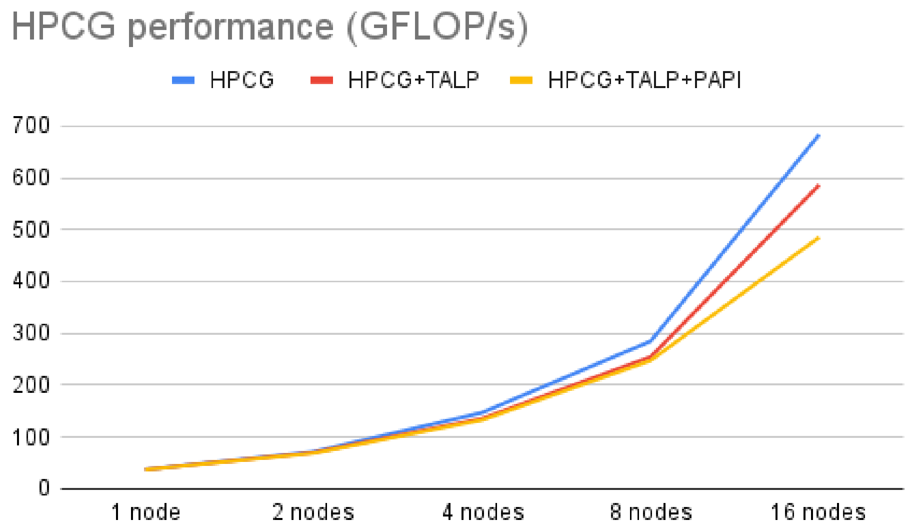


Figure 17
HPCG performance in GFLOPs in seconds in different scenarios and number of nodes used in the execution. Source: own

Each benchmark represents a different feature. LINPACK is compute-intensive, it makes a low number of MPI calls for a benchmark. The drop in scalability in the particular case of 8 nodes suggest that there is something that does not suit well with problem size, but it is not related to TALP execution because the difference of overhead between a TALP and a non-TALP execution is maintained in the bigger node cases. HPCG apart from computing it also tests data access patterns. PILS is created to simulate an imbalance in MPI tasks, but communication is barely significative. Because PAPI counters are read with every MPI call, the overhead should be bigger in applications with more communication. For the tests performed, this theory corresponds to the graphs' results, HPCG and LINPACK report a drop in performance palpable in comparison with the other benchmarks, where you can see how they run faster in plain executions but the difference between plain TALP and TALP with PAPI enabled is pretty low, even sometimes it gives a better performance the PAPI execution.

The variability in similar executions can be explained with the same reasons than the variability in measured cycles. Memory accesses, processor pre-fetching and noise from other processes make normal a variability in a program execution.

Overall, we can conclude that TALP introduces an overhead to a program execution, but that the difference between TALP with PAPI or without it depends in the communication pattern, making it hardly noticeable when the application we are running barely uses MPI.

5. Making the new information meaningful

All of these implementations will make it easier for developers and the scientific community to trace their executions. But they can see the numbers printed and the screen and not understand what that truly means or what can be going wrong.

A new section has been written and added to the User Support Team official documentation[54] called Performance Analysis Tools. This intends to introduce the user to other in-house tools, explanation of some metrics and links to webpages of entities dedicated to educating on this topic, where they will find loads of information, webinars and exercises.

6. Future development

Due to time constraints, certain features that would have been beneficial to implement were not included in the project. One such example is the ability to print all possible metrics generated by TALP in the UserPortal. Another feature that could be considered for future development is the use of GPU PAPI counters. As MareNostrum5 coming soon will feature an accelerated partition, it would be valuable to develop a tool to monitor and ensure that GPUs are being used efficiently. This would require additional research and development and may be considered as an extension to the current project.

[54]https://www.bsc.es/supportkc/

7. Conclusions

In conclusion, this thesis has presented the design, implementation, and evaluation of a library for tracing useful computation in MPI applications, which is integrated with a web-based portal for easy visualization and analysis of the captured information.

The significance of this work lies in the importance of performance analysis in high-performance computing (HPC) applications. With the increasing demands placed on HPC systems, it is crucial to have tools available to understand and optimize the performance of these applications. By providing a user-friendly and easy-to-use interface, the library and web portal developed in this thesis make performance optimization more approachable and efficient. Furthermore, with the growing concern of energy consumption in HPC, the ability to identify and eliminate unnecessary communication and computation can lead to significant energy savings, which is important to promote green computing. The library and web portal introduced in this thesis can be a significant contribution for researchers, developers and users to optimize their MPI applications, thus, achieving better performance, faster results, and energy efficiency.

The other significant aspect of this work is the dedication, attention to detail and time that went into creating a polished and professional product. The effort required to understand the subject matter, learn new technologies, and refine the system structure and code to the best of my abilities was significant. Many iterations of the scripts were written and improved over time, and extensive testing was conducted to ensure satisfaction with the final results.

A. Glossary

This glossary intends to provide definitions of concepts mentioned within the project, that require a definition for someone not familiar with HPC, an IT environment or the terms used by the software in this project, but the explanation is not long enough to make it a section inside ”Terms and concepts”.

CI/CD CI/CD stands for Continuous Integration and Continuous Deployment. It is a software development practice that aims to automate the process of building, testing, and deploying software. The goal of CI/CD is to ensure that code changes are integrated and deployed to production as quickly and smoothly as possible, while also ensuring that the software is stable and reliable.

InfluxDB InfluxDB is an open-source time series database. It is optimized for storing and querying large amounts of time-stamped data, such as metrics and events.

MySQL MySQL is a widely used open-source relational database management system (RDBMS). It is used to store and manage large amounts of structured data in a highly-scalable and reliable way

Node Single physical or virtual machine in a networked computing environment. Each node typically has its own CPU, memory, storage, and networking capabilities. They can be used as a single machine or as a set of machines that work together to perform a specific task.

GPFS GPFS (General Parallel File System) is a high-performance, parallel file system that allows multiple clients to simultaneously access and process shared data files. GPFS is designed to handle large-scale data storage needs and provide fast access to large amounts of data, making it well-suited for use in high-performance computing (HPC).

Wrapper A wrapper is a script or program that is used to interface with other software. Wrappers are commonly used to simplify the use of complex software, by providing a more user-friendly interface.

Macros When using the open-source tool Autoconf, a macro is a predefined code snippet that can be used to perform a specific task or set of tasks. Autoconf macros are typically used to check for the presence of certain features, libraries, or programs on the system where the software is being built, and to configure the software accordingly.

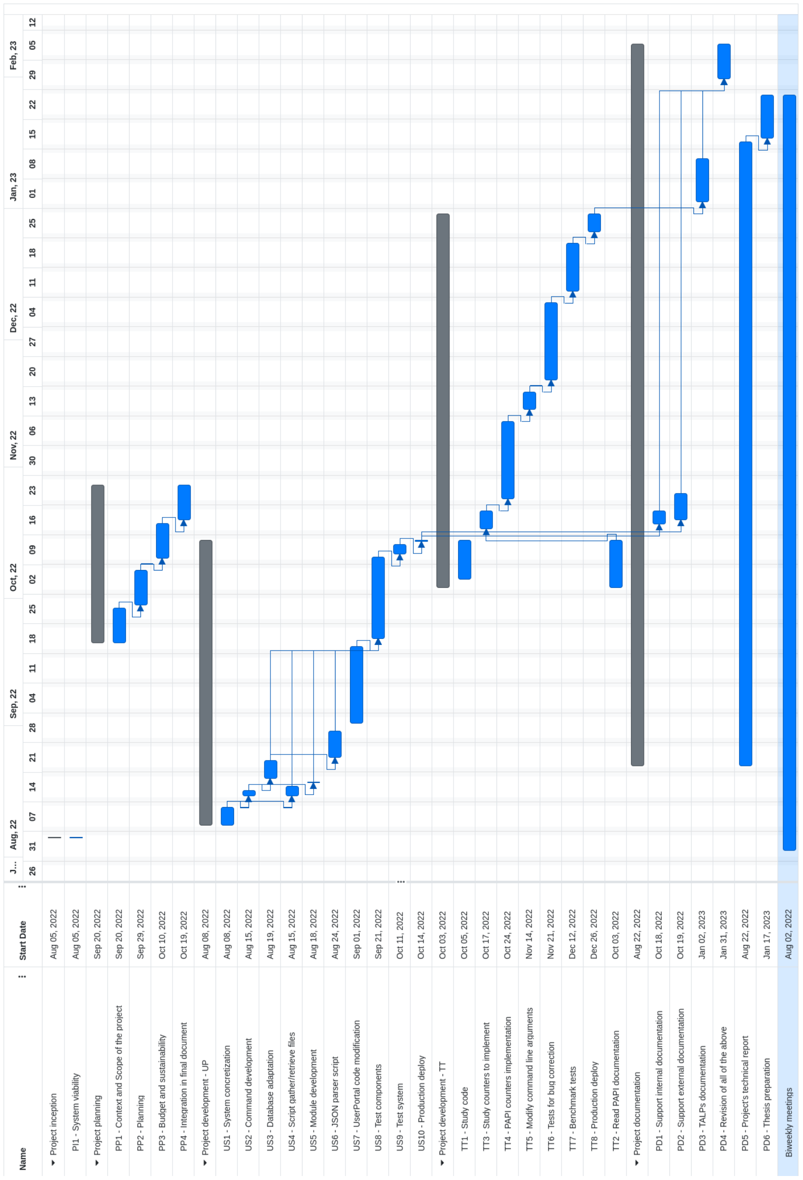
Cron Cron is a time-based job scheduler in Unix-like operating systems. It allows users to schedule scripts or programs to run automatically at specified intervals.

Module A collection of software that can be loaded and unloaded as needed. It is a way to organize software, in order to make it easy to manage different versions and dependencies, it allows you to easily switch between different software packages and versions without affecting other software on the system.

Logfile File that contains a record of events that have occurred in a system, application, or service.

KISS Keep It Simple, Stupid", is a design principle that emphasizes the importance of simplicity in software development. The principle is based on the idea that simple solutions are easier to understand, maintain, and troubleshoot than complex solutions.

B. GANTT diagram



Name	Start Date
▸ Project inception	Aug 05, 2022
PI1 - System viability	Aug 05, 2022
▸ Project planning	Sep 20, 2022
PP1 - Context and Scope of the project	Sep 20, 2022
PP2 - Planning	Sep 29, 2022
PP3 - Budget and sustainability	Oct 10, 2022
PP4 - Integration in final document	Oct 19, 2022
▸ Project development - UP	Aug 08, 2022
US1 - System concretization	Aug 08, 2022
US2 - Command development	Aug 15, 2022
US3 - Database adaptation	Aug 19, 2022
US4 - Script gather/retrieve files	Aug 15, 2022
US5 - Module development	Aug 18, 2022
US6 - JSON parser script	Aug 24, 2022
US7 - UserPortal code modification	Sep 01, 2022
US8 - Test components	Sep 21, 2022
US9 - Test system	Oct 11, 2022
US10 - Production deploy	Oct 14, 2022
▸ Project development - TT	Oct 03, 2022
TT1 - Study code	Oct 05, 2022
TT3 - Study counters to implement	Oct 17, 2022
TT4 - PAPI counters implementation	Oct 24, 2022
TT5 - Modify command line arguments	Nov 14, 2022
TT6 - Tests for bug correction	Nov 21, 2022
TT7 - Benchmark tests	Dec 12, 2022
TT8 - Production deploy	Dec 26, 2022
TT2 - Read PAPI documentation	Oct 03, 2022
▸ Project documentation	Aug 22, 2022
PD1 - Support internal documentation	Oct 18, 2022
PD2 - Support external documentation	Oct 19, 2022
PD3 - TALPs documentation	Jan 02, 2023
PD4 - Revision of all of the above	Jan 31, 2023
PD5 - Project's technical report	Aug 22, 2022
PD6 - Thesis preparation	Jan 17, 2023
Biweekly meetings	Aug 02, 2022

Figure 18
Gantt diagram for my project. Source: own

C. TALP execution diagram

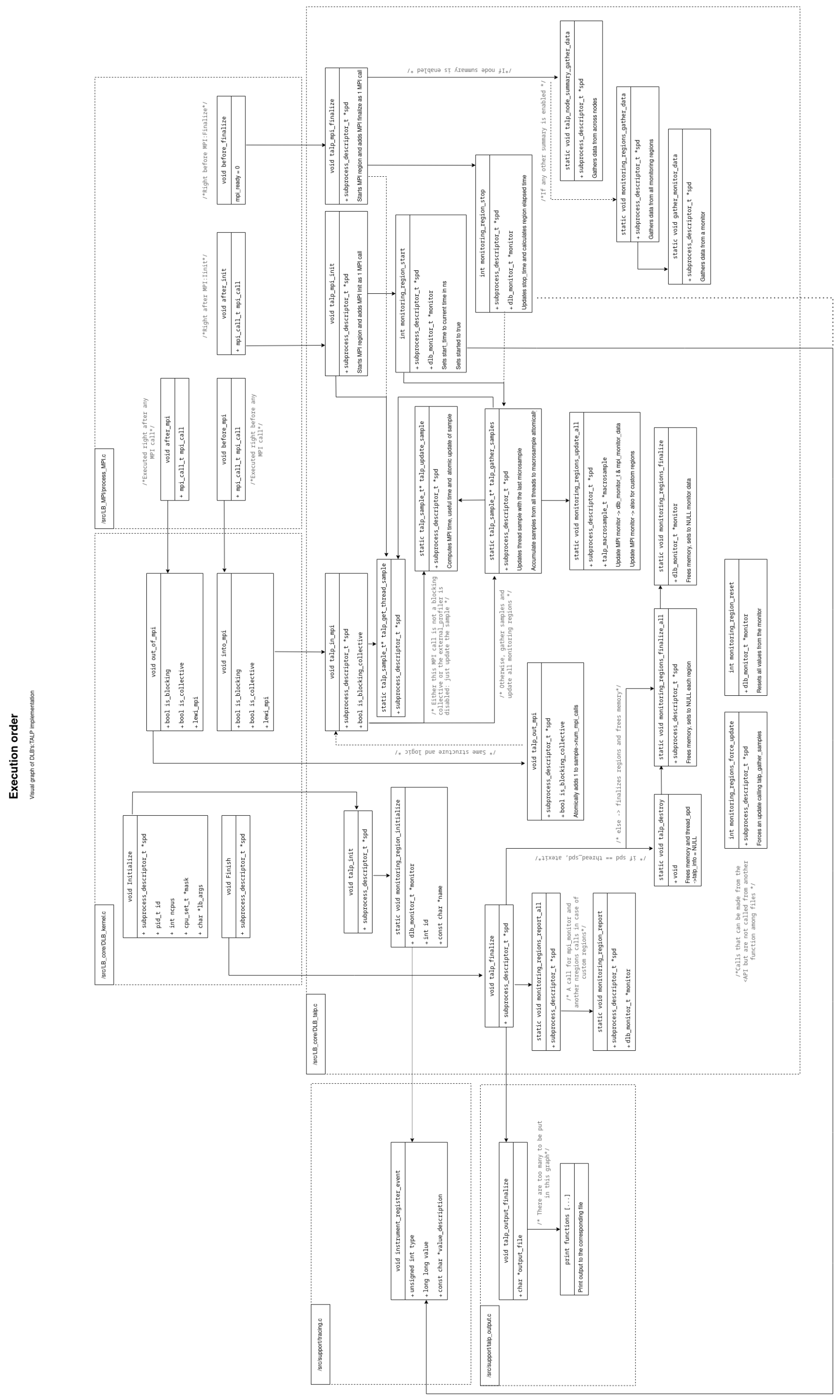


Figure 19
TALP execution diagram I have made for myself to have a better understanding of the application. Source: own

D. TALP execution diagram with PAPI

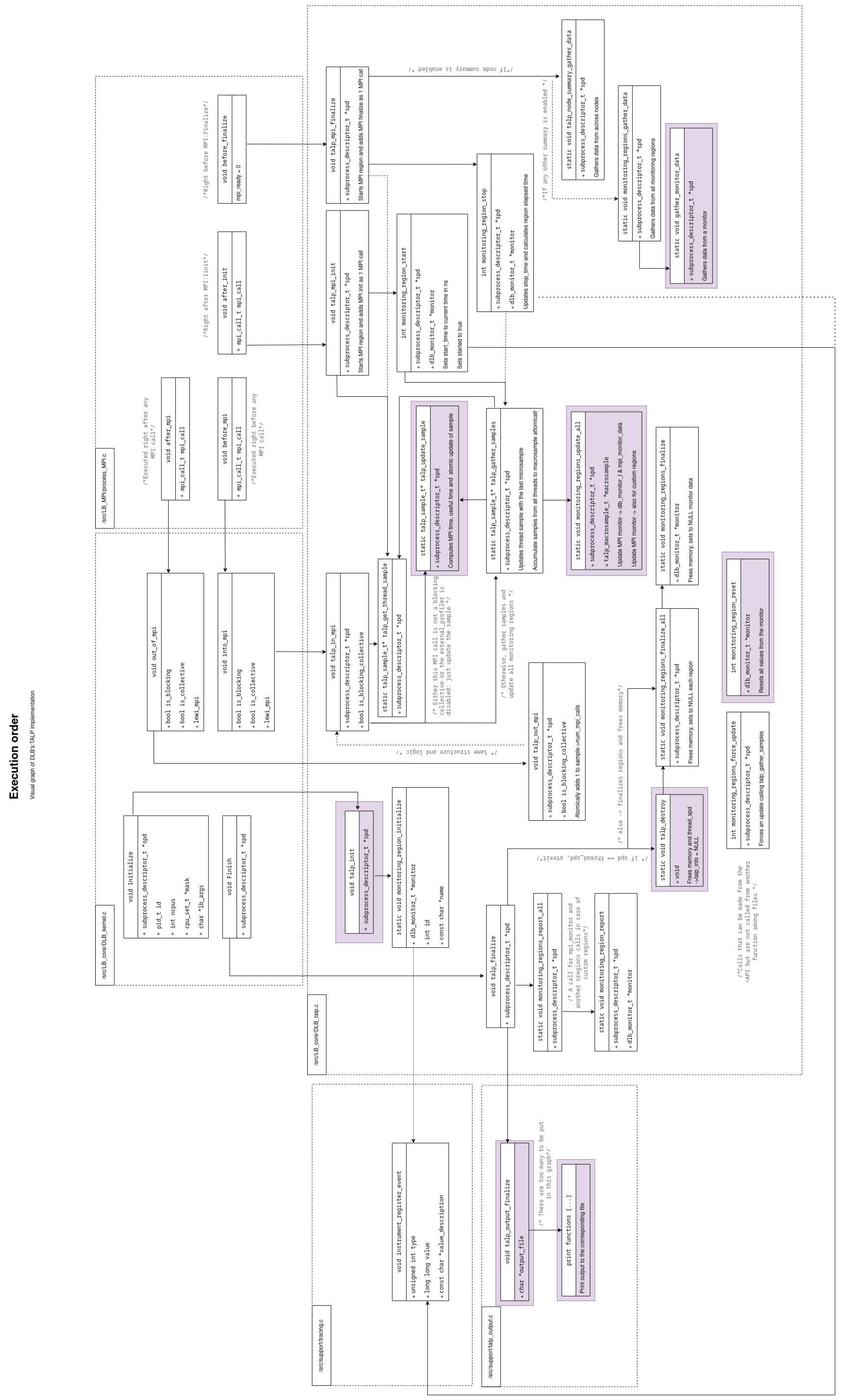


Figure 20
TALP execution diagram from before with the altered functions marked with purple Source: own

References

[1] BSC. Marenostrum 4 begins operation, 2017. https://www.bsc.es/news/bsc-news/marenostrum-4-begins-operation [Online, Accessed 25-09-2022].

[2] BSC. Bsc transparency portal, 2022. https://www.bsc.es/discover-bsc/transparency [Online, Accessed 27-09-2022].

[3] Jack Dongarra Horst Simon Martin Meuer Hans Meuer, Erich Strohmaier. Top500 ranking, 2022. https://www.top500.org/ [Online, Accessed 25-09-2022].

[4] Innovative Computing Laboratory (ICL). Papi's wiki. https://bitbucket.org/icl/papi/wiki/Home [Online, Accessed 25-09-2022].

[5] NetApp. What is high performance computing. https://www.netapp.com/data-storage/high-performance-computing/what-is-hpc/ [Online, Accessed 24-09-2022].

[6] Guillem Ramirez Miranda Victor Lopez and Marta Garcia-Gasulla. Talp: A lightweight tool to unveil parallel efficiency of large-scale ex-equations.", 2021. In PERMAVOST '21. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn [Online, Accessed 23-09-2022].

[7] Wikipedia. Message passing interface. https://en.wikipedia.org/wiki/Message_Passing_Interface [Online, Accessed 26-09-2022].

[8] Wikipedia. Mpich. https://en.wikipedia.org/wiki/MPICH [Online, Accessed 27-09-2022].

[9] Wikipedia. Openmpi. https://en.wikipedia.org/wiki/Open_MPI [Online, Accessed 27-09-2022].

[10] Wrike. Kanban guide. https://www.wrike.com/kanban-guide/ [Online, Accessed 26-09-2022].