Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Informatica

Data-driven Job Scheduling in Constraint Programming for HPC Systems

Relatore:
Chiar.ma Prof.ssa
Zeynep Kiziltan

Presentata da:
Valerio Remediani

Sessione III
Anno Accademico 2016/2017

"And when our worlds
They fall apart
When the walls come tumbling in
Though we may deserve it
It will be worth it"
[Depeche Mode - Halo]

Sommario

La presente tesi si colloca nell'ambito dei sistemi eterogenei High-Performance-Computing (da ora in poi denominati HPC), che hanno la peculiarità di utilizzare svariati tipi di risorse. In particolare, lattenzione sarà concentrata sulle tecniche di scheduling che agiscono allinterno di questi sistemi, laddove per scheduling si intendono i metodi funzionali ad elaborare i vari jobs che vengono sottomessi dagli utenti ai sistemi stessi.

La trattazione si aprirà con una prima parte di tipo teorico necessaria a delineare le premesse necessarie allapplicazione pratica ad un caso specifico che seguirà e costituirà il nucleo centrale di questa ricerca. Al fine di fornire una base metodologica, verranno quindi presentati vari tipi di scheduler, alcuni dei quali si basano sulla constraint programming, ovvero un paradigma di programmazione che prevede che le relazioni tra le variabili siano indicate sotto forma di vincoli. Per quanto riguarda, invece, la predizione della durata dei jobs forniti dagli utenti, saranno utilizzate tecniche di tipo data-driven.

Per quanto riguarda l'applicazione pratica dei suddetti metodi e concetti, verrà preso in esame EURORA, un sistema HPC di proprietà del CINECA, con sede a Bologna. Sarà, dunque, utilizzato un workload ottenuto dai log del sistema, dove sono presenti tutti i jobs sottomessi dagli utenti nell'arco cronologico di un anno, oltre a delle informazioni utili per la validazione degli scheduler che saranno meglio specificate col procedere della trattazione. E necessario premettere che, come criterio di preparazione dei jobs, i diversi tipi di scheduler adopereranno lo slowdown, i. e. una metrica comune finalizzata alla valutazione degli algoritmi di scheduling. Infatti, gli scheduler utilizzeranno tale metrica come criterio di priorit in modo da ottenere dei livelli soddisfacenti del parametro detto Quality of Service. Inoltre, saranno valutati gli impatti dei suddetti scheduler modificati sulle predizioni della durata dei jobs ottenuta grazie alle citate tecniche di data-driven.

Lintero lavoro sperimentale cos presentato è stato possibile grazie ad AccaSim, un simulatore di sistemi HPC sviluppato dal Dipartimento di Informatica - Scienza e Ingegneria (DISI) dellUniversità di Bologna. Infatti, solo in virtù della scalabilità e della customizzazione caratteristiche di questo simulatore, è stato possibile lavorare in un ambiente simulato e testare con il workload di Eurora i vari tipi di scheduler. Il fine ultimo rag-

giunto dalla presente tesi di laurea, e supportato nelle pagine a venire da dati sperimentali raccolti sul campo, consiste nel dimostrare che l'utilizzo dello slowdown, in combinazione con euristiche data-driven per la predizione della durata dei jobs, è di grande beneficio alle performance dei sistemi HPC.

Contents

List of Figures

Chapter 1

Introduction

1.1 Context

Nowdays computers have largely become part of the daily life, being an element more and more present both in young peoples entertainment and in adults professional life. There are two main types of computers : desktop computers which are designed for regular use at a single location near a desk owing to its size and power requirements and notebook computers which are portable personal computers, often with lower performances than desktop's.

In this thesis we focus on High Performance Computing (HPC) which refers to a category of computing systems that combine computing power from multiple units to deliver vastly higher performance than desktop computers can provide [2]. HPC systems have become an important instrument in the field of computational science, and they are used for a wide range of computationally intensive tasks in various fields, including quantum mechanics, weather forecasting, climate research, oil and gas exploration and molecular modeling.

In a HPC system, users submit jobs, which will form a "queue". The scheduler maintains the queue as the pool of jobs available and ,according to the scheduler policy, decides which jobs to run next among those waiting in the queue. Constraint programming (CP) has shown great success when dealing in the field of job scheduling [15][1][19].

The research in data science has shown to bring benefits when applied to the field of HPC systems, and recent works has confirmed that data science benefits scheduling decisions on HPC systems [8][9][10][11]. In this work will use a data-driven approach [13] in order to investigate whether more efficient scheduling decisions can be made.

1.2 Motivations and goals

At the state of art, a work which study CP scheduling decision in field of HPC system with the support of data science is presented in the article *Data-driven job dispatching in HPC systems* [13]. In the cited work, the authors investigate if a data-driven approach can improve the performances of five state-of-the-art scheduler methods among which an Hybrid Constraint Programming scheduler (CPH) has been used for the experiments. As a result, The previously mentioned CPH seems to be improved only by a perfect prediction and not by the data-driven prediction.

The aim of this thesis is to accept the challenge of improving the performances of the CPH scheduler and of a related scheduler also used in the work previously cited: the Priority Rule Based scheduler (PRB). Hence in this work we take into account both CPH and PRB schedulers. These performances have been obtained from the scheduling of an HPC workload in which a data-driven technique was used to predict jobs' duration.

Therefore, we continue to investigate about the results obtained by the CPH and PRB scheduler and we propose a solution to improve the schedulers performances injecting data science in the context of HPC systems. In particular the goal of the proposed solution is to show that the slowdown [20], a common metric that evaluates job scheduling algorithms, used as a criterion of job scheduling priority can lead benefits to performances in terms of jobs waiting times and system throughput. In order to proof this goal we will test the solution on the CPH and on the PRB. We have applied the same priority crierion on a different CP scheduler, the one implemented by Bridi et al. [1] for HPC systems (pure CP). We will compare the schedulers by different metrics of evaluation in order to show the improvements. All the tests are performed on the workload of a real HPC system, Eurora, a heterogeneous HPC system developed by CINECA, in Bologna. We have used python as a programming language for the testing and validation of the schedulers previously cited. All these procedures were possible thanks to Accasim, an HPC simulator for workload management [12], and thanks to the or-tools library, a set of operations research tools developed at Google.

1.3 Overview

In the next chapter, we introduce the concepts and the notations that are needed for the reader to understand this thesis. In particular, we first introduce the domain of this thesis, HPC systems and the scheduling problem, then we formally introduce CP, constraint satisfaction problem and search types. We conclude this chapter giving some information regarding data science and its application on HPC systems.

Afterwards, in the Chapter 3, will analyze recent performances obtained by the CPH and PRB schedulers over a HPC system workload, in which useful information has been predicted by the data-driven approach, in order to find its weaknesses[19] and we will propose a solution to improve their performances. In particular, we first introduce Eurora, a real HPC system, and its workload, then in Section 3.2 we present the schedulers used in the article for the experimental results. In Section 3.3 we show results obtained by the schedulers in terms of slowdown and queue size, then we will look into the observed problem of these results and finally we present a possible solution to the found problem.

In Chapter 4 we apply the proposed solution, which consists in the use of the slowdown metric as a jobs' priority criterion, over all the three schedulers taken into account. In particular Section 4.1 we present the PRB scheduler, and where we have inserted the slowdown metric in the priority criterion. Section 4.2 describes formally how the CPH works and its relative job priority replacement and Section 4.3 presents the pure CP and how we have changed its jobs' priority.

Chapter 5 shows the experimental results obtained by the schedulers taken into account. In Section 5.1 we describe the methodology used for the experiments. Section 5.2, 5.3 and 5.4 shows the results obtained by the considered schedulers with different jobs' priorities. We finally conclude this chapter with Section 5.5 showing the experimental observation.

Chapter 7 concludes the thesis, with a discussion of the obtained results and future work.

Chapter 2

Backgrounds

In this chapter, we introduce the concepts and the notations that are needed for the reader to understand this thesis. In particular, in Section 1.1 we introduce the HPC systems and expose how they are composed, in Section 1.2 we will overview the main notions of constraint programming, and then in in Section 1.3 we will talk about data science.

2.1 HPC Systems

High Performance Computing (HPC) systems refers to a category of computing systems that combine computing power from multiple units to deliver vastly higher performance than desktop computers can provide [2]. We can say that High Performance Computing most generally refers to the practice of aggregating computing power in a way that delivers much higher performance than one could get out of a typical desktop computer or notebook in order to solve large problems. HPC systems are used to perform very complex, data-intensive and resource-hungry tasks (or jobs) in a reasonably small time, which is very common today in research. For that reason HPC systems have become an important instruments in the field of computational science, and are used for a wide range of computationally intensive tasks in various fields, including quantum mechanics, weather forecasting, climate research, oil and gas exploration, molecular modelling.

Actually, following the TOP500 fastest supercomputer, the most powerful HPC system in the world is the **Sunway-TaihuLight** located at the National Supercomputing Center in Wuxi, China. Is composed by 40000 computing nodes, grouped hierarchically at multiple levels. This supercomputer uses a total of 40,960 Chinese-designed SW26010 manycore 64-bit RISC processors based on the Sunway architecture [5]. Each processor chip contains 256 processing cores, so total number CPU cores amounts to 10,649,600

across the entire system and the total memory amounts to 1.31 PB (5591 TB/s total bandwidth). Then comes the **Tianhe-2**, located in National Supercomputer Center in Guangzhou, China. Is composed by 16,000 computer node, each of them comprising two Intel Ivy Bridge Xeon processors and three Xeon Phi coprocessor chips with a total amount of 3,120,000 CPU cores.

In these super-machines a key challenge is the energy efficienty. In fact if we should notice that Tianhe-2 consumes around 17.8 MW of power dissipation to reach 33.2 PetaFlops. In the future is Expected to reach the ExaFLOP level but Exascale supercomputers built upon today's technology would led to an unsustainable power demand (hundreds of MWatts) this means that the development of new techniques and architectures aimed at improving the energy efficiency and sustainability of HPC systems has become very important.

2.1.1 HPC Architecture

Each HPC systems may have different architectures, depending on the context they are used for, now we will present a generic architecture of an HPC system.

Generally an HPC system architecture is divided in the *physical architecture* and the *software architecture*. If we read the article [5], we can have a better overview of the physical architecture, but generally it is composed of:

- **Frontend & Backend** : the frontend is composed a certain number of server through which users can interact with it. With backend of an HPC system we refer to the management part, the state of the whole HPC of the nodes and of the resources.

- **Computing Nodes** : the basic elements that form the computing system, their main work is to execute jobs, and they have the most computational power available.

- **Super Nodes** : a set of computing nodes which then connect to each other through a central switch network.

- **The Network** : is an important part of every HCP system. It generally consists of three different levels, with the central switching network at the top, super node network in the middle, and resource-sharing network at the bottom. The super node network connects all the computing nodes together. The central switching network is responsible for connections and data exchange between super nodes. The resource-sharing network is responsible of the sharing resources to the super

nodes, and provides services for I/O communication and fault tolerance of the computing nodes.

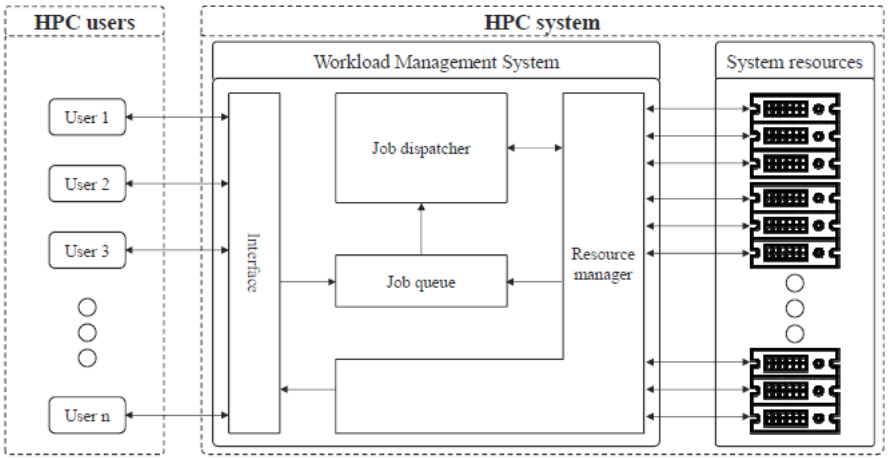

Figure 2.1: A simplified software architecture of a HPC system simulator taken from [12]

The software architecture of an HPC system is more common than the physical architecture in each HPC system. In the figure 2.1 we can see a simplified architecture of the HPC simulator which will be used in this thesis. Generally it is composed by three main parts:

- **User Interface** : generally it is a GUI or a command-line interface where the user controls the state, submites and define some attributes of the jobs, configure the HCP system and so on
- **Scheduler** : the part that decides which jobs have to start among those waiting in a queue, generally these decisions aim to complete all jobs in the shortest amount of time possible while keeping the system utilization high.
- **Allocator / Resource manager** : decides which resource has to allocate for running the jobs.

Generally, the allocator and the scheduler are components of the *dispatcher* which is a software component in HPC systems, which selects pending jobs from the queue, and allows them to start their execution. The dispatcher should be as fast as possible and not computationally intensive, since it is invoked during every process switch. It can be used in different in a *proacrive* or in a *reactive* way:

- **Proactive** is invoked when significant events like when new jobs arrive on the queue, when the system changes its state etc.

- **Reactive** is invoked at regular time intervals.
  Another important component of an HPC system is the *workload* which is a set of jobs relative to a certain time frame, that need to be dispatched according to their submission time values and resource requests. Usually the workload gives the jobs' real durations, allowing the workload to be used in simulated environments e.g. the comparisons between multiple dispatching techniques. Usually it is stored in a text-like file, with usefull informations about any single job.

2.1.2 The HPC Scheduler

In this thesis we will focus on one of the three important parts of the HPC system, the scheduler. The scheduler is a program that implements a scheduling algorithm which, given a set of requests for access to a resource (typically access to the processor by a process to be performed), establishes a temporal order for the execution of such requests, favoring those that respect certain parameters according to a certain scheduling policy, in order to optimize access to this resource and thus allow the fulfillment of the desired service,education or process.

In HPC systems, the scheduler schedules submitted jobs of the user, hence decides which waiting job has to be executed. For each scheduled job, the scheduling policy searches sequentially the nodes in an attempt to find resource available for running the job and if succeeds, it maps the job onto those nodes [13].

There are two main classes of schedulers, they could be *online* or *offline*. With *offline* scheduler we refer to a scheduler which its decisions are made prior to the running of the system, the offline type knows when a job will finish, and when new tasks will arrive. The *online* scheduler instead doesn't know when a job will finish, and when a new job will arrive, its decision are made in runtime. In this thesis we will focus on the online class of schedulers.

There are various scheduling algorithms that take into account various needs and which may be more suitable in some contexts than in others. The choice of the algorithm to use depends on five main criteria:

- **CPU utilization**: the CPU (ie the processor) must be active as much as possible, i.e. the possible "dead times" must be reduced to a minimum value.

- **Throughput**: the number of completed jobs in a certain amount of time.

- **Completion time**: the total amount of time between the submission of the job and its completion.
- **Wait time**: the time passed between its submission and its starting time.
- **Answer time**: the time that elapses between submitting the process and obtaining the first response.

In HPC systems most of the schedulers aims to minimize the wait time of all submitted jobs. Some important schedulers used in HPC systems are :

- **Shortest job first**: this scheduler gets from the queue the job which is supposed to have the shortest running time. This scheduler causes delays for the execution of the jobs which have a long running time.
- **Longest job first**: is exactly the opposite of the SJF, it get from the queue the job which is supposed to have the longest running time, causing delays for the execution of short jobs.
- **First in first out**: is a scheduling policy where the oldest job (first), or 'head' of the job queue, is processed first.
- **Priority rule-based**: is an extension of the FIFO policy (First In First Out), this scheduler sorts the set of jobs to be scheduled by certain rule, which can be changed and tuned according to the users' needs and the system type, running those with higher priority first.

After giving an illustration of the typical schedulers used in HPC systems, we present now the scheduling problem in HPC systems.

2.1.3 HPC Scheduling Problem

As said in the previous chapter, this thesis focuses on HPC systems schedulers, but every HPC system has problem related to jobs scheduling. The problem consist in the assignment of a *start time* to each submitted job in order to maintain system utilization high while keeping waiting times low. According to [6] there are two main approaches to scheduling jobs in HPC systems:

- **Queuing**: the submitted jobs join in a queue which is ordered according to a scheduling policy, e.g. FCFS or PRB. The highest prioritized job request is always the queue head. The main task of the system is the assignation to each job in the queue, free resources to requests of the waiting jobs. Therefore the scheduler checks if the job in the queue head can fit the current available resources in the

system. If it is possible, the scheduler assigns a starting time to the prioritized request. If not enough resources are available to start any of the queue heads, the system waits until enough resources become available. It can happen that more than one job can start, in that case the scheduler uses more than one criteria like queue priority or best fit (e.g. leaving least resources idled) and assigns a starting time.

- **Planning**: The planning approach consists in an assignment of start times to all requests in order to create a schedule plan for all the jobs joined in the queue without violating the systems resource constraints. For this type of scheduling approach, estimated duration are mandatory. The planning approach is the most complex and less used HPC system.

The most used approach is the queuing due to the fact that is very simple not computationally complex as the planning approach. In order to understand better the scheduling problem we now present a formal definition of the scheduling problem, according to [1].

We consider a set of jobs J = $\{j_1,...,j_n\}$. Each job has a value d_i which represents its maximal expected duration (named wall-time) and a number of jobs units u_i which refers to the number of virtual nodes required. Particularly each job unit starts and ends when the job finish.
Every job $j_i \in J$ is submitted to a specific queue q_h where $Q = \{q_1,...,q_m\}$, if we want to obtain the queue q_h we use the function $queue(j_m)$. the job i enters in the queue at time stq_i. Each queue is characterized by its expected waiting time ewt_h, which provides a rough indication of the queue priority. Waiting times larger than the ewt_h do not result in penalties for the computing center manager, but they may be an indication of poor QoS.
HPC machines are organized in a set of nodes $Nodes = \{node_1,...,node_{Nn}\}$ and a set of resources $Res = \{res_1,...,res_{Nn}\}$, like for example cores, memory, GPUs and MICs. Each node $node_j \in Nodes$ of the system has a capacity cap_{jr} for each resource $r \in Res$. Note that in case a resource is not present on a node, its capacity is zero.
Each job unit k of job i requires an amount of resource req_{ikr} for each $r \in Res$.
The HPC scheduling problem accounts for finding for each job i a start time s_i, and for each job unit k of job i the node n_j where it has to be executed.

2.2 Constraint Programming

This section is dedicated to constraint programming and the basic principles of constraint satisfaction problems. Will be presented some basic concepts and examples in order to understand better the following work.

2.2.1 Constraint Satisfaction Problem

In the every day life we are surrounded by constraints. The scheduling of events in an operating system, the scheduling crews for a flight company and the course time tabling at a university and even games like sudoku or dama deal with constraints. A mechanism to formalize and describe these real problem is to use Constraint Satisfaction Problems. We will show now a classic definition of a *Constraint Satisfaction Problem* (CSP) following [3]:

A CSP P is a triple $P = \langle X,D,C \rangle$ where:

- X is an n-tuple of variables $X = \langle x_1,x_2,...,x_n \rangle$;
- D is a corresponding n-tuple of domains $D = \langle D_1,D_2,...,D_n \rangle$ such that $x_i \in D_i$;
- C is a t-tuple of costraints $C = \langle C_1,C_2,...,C_n \rangle$;

Each variable X_i is associated with a domain D_i of possible values that can be assigned to the variable. A constraint C_i specifies, for a given set X_i of argument variables, which values for those variables together "satisfy" the constraint where values for a variable are chosen only from the corresponding domain. Formally speaking a constraint C_i is a pair $\langle R_{s_j},S_j \rangle$ where R_{s_j} is a relation on the variables in $S_i = scope(C_i)$. In other words, R_i is a subset of the Cartesian product of the domains of the variables in S_i.
A solution to the CSP L is an n-tuple $A = \langle a_1,a_2,...,a_n \rangle$ where a_i *in* D_i and each C_j is satisfied in that R_{S_j} holds on the project of A onto the scope S_j. An assignment that does not violate any constraints is called a *consistent* or *legal* assignment. A complete assignment is one in which every variable is mentioned, and a solution to a CSP is a complete assignment that satisfies all the constraints. Some CSPs also require a solution that maximizes an objective function.

We show now an example of a game which can be modeled through CSP, the n-queens problem. This problem consist of placing $n$queens on an N by N chess board, so that they do not attack each other, i.e. on every row, column or diagonal, there is only one queen exists [4]. In this case we consider 8 queens, so we have an 8 by 8 chess board. A solution for the *8*-queens is depicted in figure 2.2 where no queen on the board is able to attack another one. Now we give a related model to the N-queen problem.

First of all we have to choose the variables and the domain , then we will elaborate the constraints:

- **Variables** : 8 variables X_i, which represent a queen on a row i;
- **Domain** : The domain of each variable is: $D_i = \{1, 2, ..., 8\}$, giving the possible columns that queen i can be placed;

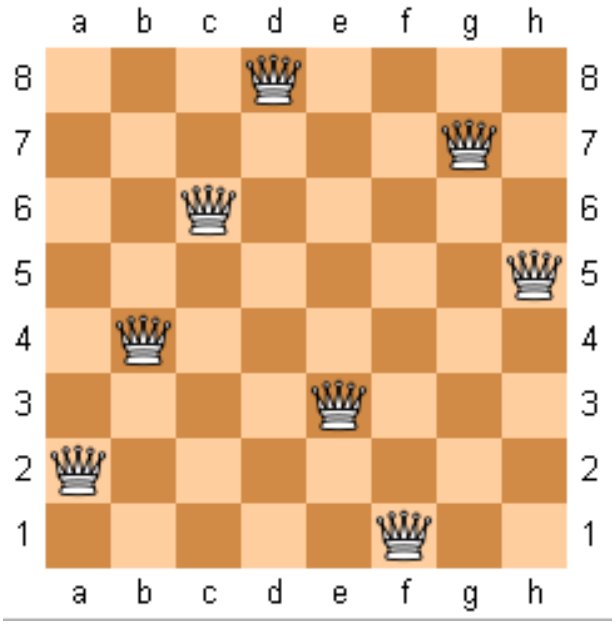

Figure 2.2: Explicit solution for 8 queens problem

- **Constraints** : $\forall$ i,j with $i \neq j$ and $1 \leq i \leq j \leq 8$, we have three different type of constraints :

  1. $x_i \neq x_j$ (two queens can't be placed in the same row)
  2. $i - j \neq x_i - x_j$ (no other queen can be placed on one of the two diagonals)
  3. $j - i \neq x_j - x_i$ (no other queen can be placed on other one of the two diagonals)

After giving a simple introduction about CSP, we introduce now constraint programming which an important topic that must be known in order to understand this thesis.

2.2.2 Constraint Programming

In computer science *constraint programming* is a powerful paradigm for solving combinatorial search problems that draws on a wide range of techniques from artificial intelligence, operations research, algorithms, graph theory and elsewhere. The constraints differ from the primitives normally defined by the other programming languages due to the fact that they do not specify single actions to be executed step by step, but rather limit themselves to specifying the properties to be provided with the solution to be found. The constraints used can be of various types: those based on constraint satisfaction problem (explained in the previous section), those that can be solved using the

Simplex Algorithm algorithm and others.

Nowadays constraint programming is currently applied with success to many domains:

- Bioinformatics
- Network like electricity, water, oil network
- Scheduling
- Vehicle routing
- Planning
- Configuration

The basic idea in constraint programming is that the user states the constraints and a general purpose and a constraint solver is used to solve them. Constraints are just relations, and a constraint satisfaction problem (CSP) states which relations should hold among the given decision variables [3]. A Constraint solvers is the tool take a real-world problem like a scheduling problem or a planning problem, represented in terms of decision variables and constraints, and try to find an assignment to all the variables that satisfy the constraints [3].

2.2.3 Costraint Solver and Search Types

A *constraint solver* searches the solution by exploring the solution space using different search techniques. A search can be *complete* or *in-complete*. With a complete algorithm is guaranteed that a solution will be found if it exists, otherwise it will prove the unsatisfiability, or

finds an optimal solution. An in-complete algorithm instead doesn't assure to find a solution. The most typical search is the local-search which is an in-complete algorithm and the backtracking search which is a complete algorithm.

The *local-search* is an informed search algorithm that can solve a problem by processing only in a subset of the search space. Starting from a node, the local search analysis identifies the solution to the problem by analyzing only the nodes adjacent to the current one. Local research belongs to the category of metaheuristic algorithms. A difference in traditional search algorithms is that the local search does not analyze multiple paths in the search space. Thanks to this analysis, the search algorithms have been used, in particular, in very large research spaces and in networked ones. Local research is based on the premise that the improvement of a solution lies in the immediate neighborhood

of the latter. Given any heuristic solution (current node) local research verifies the existence of better solutions in the set of the closest solutions (neighboring nodes). For example, a local search algorithm can be used to solve the traveling salesman problem [14]. Local search techniques allow us to achieve results both in the search for solutions to a problem (problem solving) and in optimization.

The *backtracking search* enumerates all the possibilities solutions and discards those that do not meet the constraints. A classical technique of the backtracking search is about to explore and keep track of all the previously visited nodes, so you can go back in time that contained an unexplored path in case the research in the current branch did not succeed. A backtracking application is in the chess game, which generates all the possible moves for a depth of N moves starting from the current one and then backtracking the various alternatives, eventually selecting the best one. The backtracking algorithm has an exponential complexity, so it is not very efficient with problems that are not NP-complete. However, the algorithm integrates heuristics that reduce complexity.

2.2.4 Constraint Programming in Scheduling Problems

Constraint programming techniques achieved great success when dealing with scheduling problems, indeed, thanks to its expressive and flexible language, is possible to implement powerful algorithms to quickly find good quality solutions [15][1][19].

Constraint programming, as previously said, is a well known paradigm to solve NP-hard problems by efficiently exploring the solution space for optimizing one or more objective function. This technique, however, have seldon been used in HPC facilities as it is computational expensive and this incompatible with intrinsic on-line nature of HPC job schedulers, but there are some CP schedulers which contradicts this claim [1][19].

In a CP scheduler is possible to define different objective function, depending on the used context. Commonly, a way of modeling an objective function, is simply by introducing a variable criterion that is constrained to be equal to the value of the objective function. Although the minimization of the makespan, i.e., the end time of the schedule, is commonly used, other criteria are of great practical interest e.g., the sum of setup times or costs, the number of late activities, the maximal or average tardiness or earliness, storage costs, alternative costs, the peak or average resource utilization, etc.[3]

Constraint programming offers various type of constraints, some of them are used in scheduling problems. An important constraint of CP to model a scheduler is the *cumulative constraint*. We give now a formal definition of this constraint and we will see its application in Chapter 4.

We are given a collection $T = t_1, \ldots, t_n$ of tasks, such that each task t_i is associated with four variables:

- its release time r_i which is the earliest time at which it can begin executing,
- its deadline d_i, which is the time by which it must complete
- its processing time p_i which is the amount of time it takes to complete,
- its capacity requirement c_i which is the capacity of the resource that it takes up while it executes,
- and a capacity variable C of the resource

A solution is a schedule, i.e., a starting time s_i for each task t_i such that $r_i \leq s_i \leq d_i - p_i$ (the task completes before its deadline), and in addition:

$$\forall u \sum_{i|s_i \leq u \leq s_i + p_i} c_i \leq C$$

i.e., at any time unit u, the capacity of the resource is not exceeded. Note that the starting times s_i are auxiliary variables; instead of s_i we reason about the release times r_i and deadlines d_i [3].

Another important constraint for job scheduling is the *alternative constraints*. The constraint $alternative(a_0, \{a_1, ..., a_n\})$ models an exclusive alternative between $\{a_1, ..., a_n\}$. If time-interval a_0 is executed then exactly one of time-intervals $\{a_1, ..., a_n\}$ is executed and a_0 starts and ends together with this chosen one. Time-interval a_0 is not executed if and only if none of time-intervals $\{a_1, ..., a_n\}$ is executed. This constraint can be usefull i.e. when a job is composed of two operation a b that can be executed either in series or in parallel and this is modelled by two alternatives $alternative(a, \{a_1, ..., a_n\})$ and $alternative(b, \{b_1, ..., b_n\})$ [16].

In the next section we will see how important is data science in the field of HPC systems.

2.3 Data Science

Data science is a field that must be known in order to understand this thesis, so now we will present what is data science and how it can be used in the scheduling problem.

Data science, also known as data-driven science, is a field of scientific methods used to extract knowledge from data in various structured or unstructured forms [7]. These methods are based on techniques from various disciplines, mainly from mathematics, statistics, information science, and computer science, especially in the following subdomains: artificial intelligence (or machine learning), databases and data visualization. These data-driven approaches are particularly useful when user data can be stored for longer periods of time, which is increasingly feasible through modern Big Data tools techniques and they are mostly used in communication, networking, automation and mechanics. Data-Driven is a new way of thinking, enabled by machine learning. With data-driven it is possible to implement an algorithm that can spot connections and correlations that we may not even know to suspect.

2.3.1 Data Science in HPC Systems

In High Performance Computing (HPC) systems, data science is particularly useful when user data can be stored for longer periods of time, which is increasingly feasible through modern Big Data tools and techniques [13]. One of the main challenges in HPC systems is to ensure that applications continue to achieve high throughput and complete the task despite hardware and software failures [8]. Thus, it is beneficial for the scheduler to predict the status of each job and use this information to make better scheduling decisions on workload management, provide users with feedback on the completion of their worktime, and execute proactive fault-management actions like checkpoint initiated by the system [9]. In literature we can see important works related to data prediction of job's attributes such as the job duration, job demands.

Chen et al. [9] propose a prediction technique for HPC systems that makes use of the system logs to predict job residual times. The prediction is obtained by the usage of *Hidden Markov Models* (HMM) to learn the characteristics of log messages and use them to predict job residual times. The proposed HMM approach can predict 75% of jobs within 200 seconds of error and the HMM used on short jobs (i.e., jobs that run for one hour or less), predicted on job termination in 10 minutes has a 1% false positive rate, and an overall accuracy of 93%.

Matsunaga et al. [10] propose machine learning techniques for predicting spatiotemporal utilization of resources by applications. In particular this work proposes an extension of an existing classification tree algorithm, called *Predicting Query Runtime* (PQR) and it yields the best average percentage error, predicting execution time, memory and disk consumption for two bioinformatics applications, BLAST and RAxML, deployed on scenarios that differ in system and usage.

Sirbu et al. [11] present a support vector machine mode to predict the power con-

sumption of jobs. The prediction derives from user history and authors show that when enough data is available, high performance can be achieved.

Overall, the prediction of job attributes is a useful information in scheduling decisions and knowing them at job submission time can be very important for a scheduling algorithm. In this work we will use a data driven approach to predict job duration that relies on user histories, similar to [11]. This prediction is taken into account in order to estimate the impact of these techniques on system throughput, in terms of waiting times and job queue size, in the scope of heterogeneous HPC systems, more difficult to manage than their counterparts homogeneous. We will discuss about this data-driven approach in the following chapter.

Chapter 3

Data-driven Job Dispatching in HPC Systems

In this chapter will talk about the article "Data-driven job dispatching in HPC systems" [13], we will see the problem for which the following thesis is elaborated. The chapter is structured as follows: in the Section 3.1 we have a description of the Eurora system, then in Section 3.2 we will list the scheduler used in this article in the last Section, the 3.3, we analyze the results of the experiments, the encountered problem and a possible solution.

The considered article focuses on HPC systems, in particular on job dispatching strategies. As we said in the previous chapter, these strategies are becoming critical for keeping the system utilization high and maintaining low the job waiting time. Furthermore a data-driven approach is used in the log data produced by an HPC-system in order to investigate if it is possible to have better dispatching decisions. The effect of this prediction used on job duration is analyzed. The logs of the HPC-system come from Eurora, which is a hybrid HPC. The prediction of the job duration are tested on a simulator, *Accasim*, which is developed at the University of Bologna, Italy, at the Department of Computer Science and Engineering (DISI), and its main contributors are Cristian Galleguillos, Zeynep Kiziltan and Alessio Netti. We will analyze this simulator in the last chapter, from now we just assume that the simulator has the same resources of the Eurora System.

Figure 3.1: A picture of the Eurora system. Image taken from sito di eurora

3.1 The Eurora System

Eurora is a prototype HPC system built in 2013 by CINECA in Bologna, Italy, and we can see a real picture of the system in Figure 3.1. It is a small-scale heterogeneous HPC system with an hybrid architecture and the main ability of this system is its low power consumption, in fact has reached the first position in the Green500 list of Top500, which ranks the most efficient HPC systems worldwide, in July 2013.
Eurora's architecture is a hybrid cluster based on 64 nodes, half of them host 2 powerful NViea GPUs (NVIDIA TESLA K20), meanwhile the other half is equipped with 2 Intel MIC accelerators. Tesla GPUs are parallel accelerators based on the NVIDIA CUDA parallel computing platform.
Tesla GPUs have been specifically designed for maximum energy efficiency, high performance computing, computational science and supercomputing, and offer significantly better performance with scientific and commercial applications than CPU-only systems.
MIC accellerators are *INTEL Xeon Phi* which is a bootable host processor that offers a high level of parallelism and vectorization to support the most demanding HPC applications. The integrated and energy-efficient architecture offers much more processing per unit of energy consumed than comparable platforms, to allow the reduction of overall management costs. Each node has 16 GB of RAM memory. These node are dedicated exclusively for computation, with the user interface being managed by a separate node.
The *operative system* of the nodes is Linux CentOS 6.3 distribution, and the workload management system is Portable Batch System (PBS), which employs various heuristics for optimal throughput and resource management. Eurora has been used by italian

scientist for simulation studies from different fields and for that reason, its workload is heterogeneous.
In many HPC systems, users can define a wall-time value, an estimated duration of the job, and when users fail to provide one, the HPC system uses a default value. In Eurora, users can specify the queue type based on the job type. Eurora uses three different queues for job dispatching:

- **debug** queue: in the debug queue are intended jobs that are used for debug purposes, with low execution time.
- **parallel** queue:In the parallel queue instead are intended ordinary jobs that can take at most 6 hours of execution.
- **longpar** queue: the longpar queue is designed for long jobs, with a execution of at most 24 hours, that are to be scheduled during the night.

3.1.1 The Eurora Workload

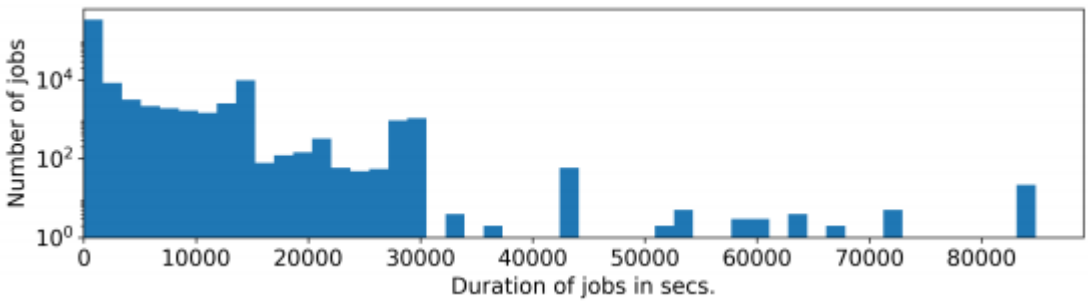

Figure 3.2: Distribution of job durations on Eurora. Image taken from [13].

The workload of Eurora includes logs for over 300000 submitted jobs between March 2014 and August 2015. In there logs, we have information for each job about the submission, the start time, the end time, the queue type, the wall-time, user and job name, used resources and their allocation of the various nodes. As mentioned earlier, in Eurora there are GPU and MIC accelerators, so we have information of how many GPU and MIC a single job has requested for its execution. Furthermore it is not possible for a job to request both GPU and MIC accelerators, because a node may have only one of these available at a time, and PBS does not allow heterogeneous job unit requests. For this article the jobs of the Eurora's workload are divided in three different type based on their duration:

- **Short job** are the jobs which their execution time do not exceed 1 hour.

- **Medium job**: are the jobs that have an execution time between 1 and 12 hours.
- **Long job**: are the jobs with an execution time exceeding 12 hours.

In Figure 3.2 we see the high frequency of short jobs, while the frequency of medium and long jobs is fewer. In terms of percentages, 93.15% of jobs fall into the short class (the vast majority), 6.82% into the medium class and only 0.03% into the long class.
The Eurora system is composed by three job queues, each with a different expected waiting time depending on the job purpose. This value, defined as ewt_q, is assumed to be of 1 hour for the debug queue, of 6 hours for the parallel queue, and of 24 hours for the longpar queue.
There are three different workload types based on estimations of job duration, we now describe these variants:

- The *wall time* is a job duration estimation and the worst results are expected from this variant, due to its severe over-estimation.
- The *real time* is a job duration estimation and we expect the best results from this variant, because is exactly how much time a job will take to complete his task.
- The *data-driven estimation* is a job duration estimation computed from the workload.

The last type of job duration estimation is calculated by a simple heuristic, we will describe the latter in the next section.

3.1.2 Job Duration Prediction

The heuristic of the article builds a *job profile* from the available workload. The job profile include information about job name, queue name, user-declared wall-time and the number of resource of each type (CPU,GPU,MIC,nodes). Generally the prediction is based on the observation of past jobs that are similar or same to the current one, and get their duration as an estimation. In particular the authors of the article have noticed that jobs with similar profiles have approximately the same duration for long periods of time, whereby there is a temporal locality of job durations. The cause of this temporal locality arises from the fact that in a certain time frame, it is possible that a job is repeated many times for debugging purposes, or with different parameter combinations, usually employing the same input data. Then at some point, the duration changes to a new set of values and remains again stable for some time. This behaviour could be due for instance because the user first test the code with short run (debug) then decides to run the real simulation which may take more time.

The heuristic consists in a set of consecutive rules, we now show the set starting from the most important:

1. At first a full profile match is serched in the user history;
2. If a full profile does not exist, a match is searched allowing the job names to have just a prefix in common;
3. If the previous searches are unsuccessful, a match is searched, allowing only the resources use to differ;
4. If the rule No. 1 and 2 fails, a match is searched, allowing not only the resources that can differ but also the job name like in the second rule;
5. If none of these rules give a match, a search is performed by using only the job name, or at least its prefix;
6. If all rules fail, the jobs wall time is used as an estimation.

The authors also used machine learning to predict job duration but the results were not satisfactory, so the simple heuristic, as described above, provided much better performance. In the next section we will list the schedulers used for the experimental results of the article.

3.2 Schedulers Used

The schedulers used in this article with the simulator Accasim are five, we now give information about them, some schedulers were introduced in chapter 2.

Shortest job first, longest job first (SJF) and (LJF) use the estimated duration in the scheduling time, the simulator maintains jobs in a sorted list in ascending or descending order, depending if we use SJF or LJF, and map the shortest job (or the longest) to a resource.

EASY-Backfilling is very popular in many commercial dispatchers. The main scope of the Backfilling is to improve the FCFS (first come first served) by increasing the utilization of the system resources and by decreasing the average waiting time of the job in the queue of the simulator. Different variants of the basic backfilling approach have been proposed, in this case the EASY version. In this version only the first job in the submission queue is allowed to reserve the resources it needs. This approach is more aggressive because it increases resource utilization, even if it does not guarantee that a

job is not delayed by another one submitted later [17].

Priority rule-based is a generalization of the FCFS, SJF and LJF algorithms. The dispatcher sort the set of jobs to be scheduled by a certain priority which can be defined by the user, running those with higher priorities first. In this case the rule is to give priority to jobs that have urgency in leaving the queue. This urgency is calculated by the ratio between the waiting time and the expected waiting time. Therefore, jobs that exceed or are closed to surpass their expected waiting time have priority over the jobs that still could wait in the queue, hence jobs that have requested less results and have short duration have further priority. This scheduler algorithm and the next one will be analyzed specifically in the next chapter because are objects of this thesis.

Hybrid constraint programming method (CPH) is a recent scheduler based on constraint programming firstly introduced by Bartolini [18] which is able to outperform PRB methods, then Borghesi et al. in [19] implemented a new CP scheduler more scalable, combining CP and a heuristic algorithm. In this article the Borghesi's version is used. The CPH scheduler is composed of two stages:

- the first stage consists in the job scheduling using CP and as objective function we have the minimization of the total waiting time. The model for the scheduling process is relaxed which considers each resource type as one unique resource. The CPH model is solved with a custom search strategy guided by a branching heuristic using the scheduling policy of PRB.
- the second stage corresponds to the resource allocation, the algorithm checks if there are some available nodes, if they exist, a job is mapped to a node and then it will be dispatched, otherwise it will be postponed.

We have now listed the scheduling algorithms which are used in the experimental part, in the next section we will see the results of these schedulers used in the Accasim simulator with the Eurora's workload.

3.3 Experimental Results

As mentioned before, this section will shows the experimental results of the article.
All the tests and the Accasim simulator were run on a CentOS machine equipped with Intel Xeon CPU E5-2640 Processor and 15 GB of RAM. The dispatching methods are the five previously introduced, together with three estimations of job duration: prediction based on the wall-time (W), based on data-driven prediction (D), and real duration (R). Therefore for each dispatching methods we will have three test results, resulting in 15 combinations.

3.3.1 Slowdown

The metric used to compare the schedulers is the *job slowdown* which is a common metric that evaluates job scheduling algorithms [20]. In order to understand what slowdown is, we define with T_r the running time, during which the job is actually running on a processing node, and the waiting time T_w, in which the job is waiting to be scheduled or for some event such as I/O. The slowdown (also called expansion factor) is the response time normalized by the running time:

$$slowdown = \frac{T_w + T_r}{T_r} \tag{3.1}$$

From the Equation 3.1 can be deduced that jobs with a long running time are less susceptible to high waiting times, jobs with low running time instead are more susceptible to high waiting times, and the slowdown will grow very rapidly as the waiting times increase. Generally if a scheduler keeps slowdown values low, it is an efficient scheduler, but we have to remember that slowdown has a big impact on short jobs mainly, so it often depends on the workload of the system. We now see the results using the slowdown as a metric to evaluate the dispatchers. The boxplot method, which allows to see more than one attribute, was used to the evalualtion: minimum an maximum values are represented with a top and a bottom line, the coloured box represent the 3rd quartiles, the median is represented by an horizontal line within the box and the mean by a green triangle.

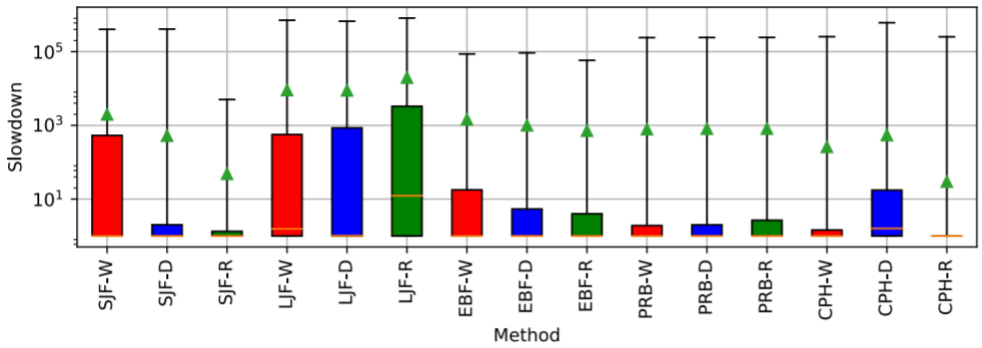

Figure 3.3: Distribution of job slowdown for each method, image taken from [13]

Figure 3.3 depicts the distribution of slowdown achieved by each scheduler algorithm with each prediction type. Jobs with slowdown equal to 1 are excluded in the figure because they are scheduled as soon as they enter the queue. The methods which have archived best performance with the least effective prediction (walltime) are PRB and CPH, while the worst performance is archived by LJF and SJF most likely because are simple methods while the others are more sophisticated. If we consider the real job duration, not all the schedulers show a clear benefit. The slowdown of LJF has a significant

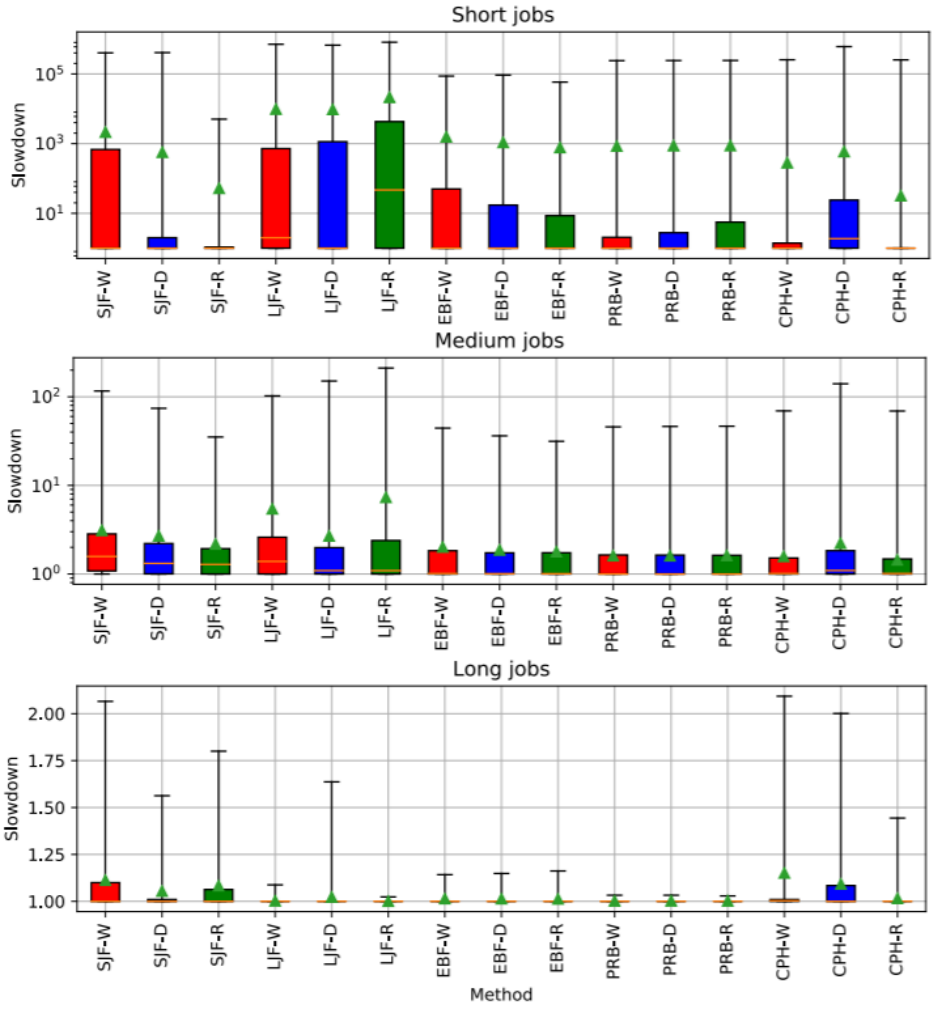

Figure 3.4: Distribution of job slowdown for short, medium and long jobs for each method, image taken from [13]

increase with real duration, instead SJF, CPH, EBF have a decrease. This behaviour can be caused because sometimes, dispatching methods try to find a suboptimal decision that is often compensated by an underestimation of job duration so that a perfect prediction is not anymore available. If we look at the data-driven prediction, it is supposed that slowdown values are between the walltime and the real duration. Most of the methods reflect this characteristic like the SJF and EBF, with LJF both real duration and data-driven prediction worsen the result. PRB which is not achieving a clear benefit with the real duration, does not benefit from the data-driven prediction either. As we can see, the only one dispatcher which is not getting benefit from the data-driven prediction

is the CPH, authors of the article believe that the data-driven prediction, as opposed to walltime and real duration, may sometimes underestimate the duration of a job. An important result is achieved by SJF which, with its simple strategy, come very close to CPH and PRB.

Figure 3.4 shows a slowdown distribution of short, medium and long jobs. The distribution of short jobs is almost similar to the Figure 3.3, this is due to the fact that the workload is composed by 93.15% of jobs with short duration. In the medium job distribution we can see some smaller differences, but not so relevant. In long jobs distribution, the schedulers seem to be quite comparable, with some high values in CPH and SJF.

3.3.2 Queue size

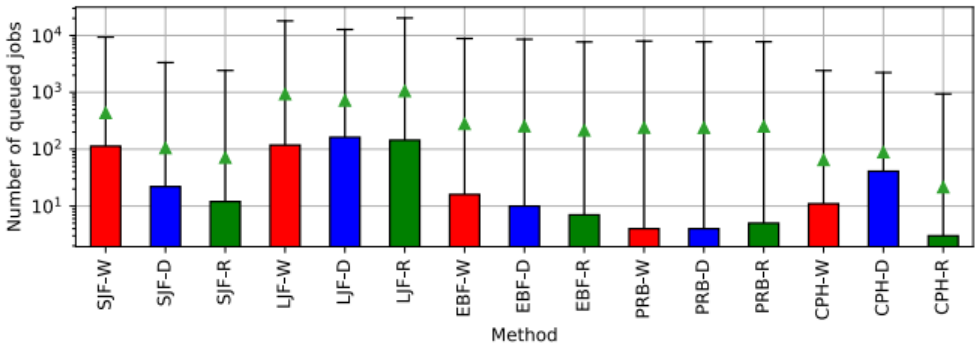

Figure 3.5: Distribution of number of jobs waiting at every second, for each method, image taken from [13]

The queue size is an highly descriptive metric of schedulers performance and is computed when the dispatcher is invoked to schedule a new job. The equation of the queue size is the following:

$$qs_t = |Q| \tag{3.2}$$

In the equation, qs correspond to the number of the jobs in the queue Q (its size) after dispatching has been performed. This metric is computed for every time step t in which the dispatcher is invoked. In the case of this article, the time t is equal to one second.
In figure 3.5 shown the distribution of number of jobs waiting at every second, the effect of the system is similar to the performance measured by slowdown. In particular CPH and LJF are not improved by the data-driven prediction while SJF and EBF are improved with both data-driven and real duration, and PRB shows no difference.

3.3.3 Observed Problem

The main problem as we can understand is that the data-driven duration prediction of a job is not getting relevant results over PRB and CPH scheduler. We are focusing on these schedulers because they have achieved the lowest results over the data-driven job duration, except for the LJF which is understandable because since the latter gives priority to long jobs and most of the jobs in the workload fall into the short class.

If we look back at figure 3.3 which is distribution of job slowdown for each method and for all the jobs, we can see that the box of the CPH performance with D (data-driven approach) is higher than CPH-W and CPH-R.
Statistically the box represents the difference between the 1st and the 3rd quartile which is the *interquartile range*, used to indicate the variability around the Median and covers the scores falling within the middle two quartiles (center 50%) of the distribution. Therefore most of the slowdown values of CPH-D are more higher than CPH with job walltime duration and than CPH with real job duration.
Overall CPH, which is the most sophisticated algorithm in this experiment, does not benefit from the author's heuristic, another proof is in the figure 3.5 where the boxplot of CPH-D, which in this case represent the queue size values, is the most higher between CPH-W and CPH-R.
Another observed problem is that PRB does not show benefits from the data-driven job duration, indeed, as we can see in figure 3.3, the performances of PRB-D decrease if we compare them to the performances achieved by PRB-W.

Furthermore we have observed that the cause of these results are due to the fact that PRB and CPH are using the expected waiting time to give priority to jobs queue, but this value is a fixed value and never change during the scheduling period; in other words the expected waiting time does not exploit well the dynamicity of the scheduling process.

This observed problems is the main purpose of the work of this thesis and in the next subsection we will show a proposed solution to get CPH and PRB benefit form the data-driven heuristic.

3.3.4 Proposed Solution

Analyzing the problem, we find out that a possible solution is to replace the expected waiting time, which is used in the scheduling priority, with the slowdown metric (Equation 4.12) in both CPH and PRB schedulers. Therefore jobs that has an high slowdown value have priority over the jobs with a low slowdown value. In other words, we are giving weights to each job waiting in the queue, using the slowdown metric. Hence we want investigate how slowdown metric can improve PRB and CPH schedulers to exploit

better the data driven job duration. The solution we proposed is more dynamic and we think is better instead of using fixed values in the schedulers priority.

In particular we want to replace the priorities based on the expected waiting time with priorities based on the slowdown metric:

- in the *priority rule* of the PRB so that the latter sorts the jobs to be scheduled in decreasing order of the jobs' slowdown i.e. if a job is waiting more than its duration, it must be scheduled as soon as possible.
- in the *objective function* of CPH, so that the algorithm minimizes all the slowdown values of the jobs waiting in queue.
- in the CPH *custom search strategy* so that the algorithm's search is based on jobs slowdown values.

With these replacements, we want to show that CPH and PRB will become more performing, in terms of slowdown, queuesize and waiting time with the data-driven prediction. Furthermore we will apply this new priority on a different CP scheduler, introduced and implemented by Bridi et al. [1] in "A Constraint Programming Scheduler for Heterogeneous High-Performance Computing Machines" and, in order to distinguish the latter from the CPH, we let pure CP denote this scheduler. Both CPH and pure CP are based on constraint programming but differs in some constraints. In the next chapter we will show how the actual jobs' priority based on the expected waiting time is replaced with the new proposed jobs' priority on the CPH, PRB and on the pure CP.

Chapter 4

Job Scheduling Decisions based on Slowdown

In this chapter we will present and describe the scheduling methods of PRB,CPH,pure CP, and their relative jobs priority and after we will apply the solution previously introduced.

In Section 4.1 we will discuss about PRB, showing its pseudocode and its scheduling priority. In Sections 4.2 we will then discuss about the CPH scheduler, analyzing the search and the objective function and in the same way, in Section 4.3 we will discuss about the pure CP.

4.1 PRB Scheduler Overview

The PRB, known in the literature as *priority Rules* and already introduced in the previous chapter is a scheduler which gives priority to jobs that follow a certain rule. In this thesis has been taken into consideration a variant of the PRB, implemented by A. Borghesi et al. [19]. The variant of this implementation is to order a set of jobs which have to be scheduled in a ordered list by assigning a priority to each job. The order is based on a set of rules which determinate the priority of each job. The jobs are ordered with respect to their expected waiting time and, as a break ties, is considered the job demand which is calculated by the estimated duration multiplied by the job requirements. We can understand that jobs that are expected to wait less in the queue have higher priority, therefore short jobs are preferred over long jobs.

In order to describe how the PRB is implemented we now show the pseudocode of the

algorithm, taken from [19]

Algorithm 1: PRB algorithm

```
1  time ← 0 ;
2  startTimes ⟵ ∅ ;
3  endTimes ⟵ ∅ ;
4  runningJobs ⟵ ∅ ;
5  orderByRules(R) ;
6  orderByRules(J) ;
7  while J ≠ ∅ do
8      for j ∈ J do
9          canBeMapped ← true ;
10         for rn_i ∈ j do
11             for r ∈ R do
12                 if checkAvailability (rn_i, r) then
13                     updateUsage (rn_i, R) ;
14                     break ;
15                 else
16                     canBeMapped ← false
17         if canBeMapped = true then
18             J ⟵ J − {j} ;
19             runningJobs = runningJobs ∪ {j} ;
20             startTimes (j) ← time ;
21             endTimes (j) ← time + duration (j) ;
22         else
23             undoUpdates (j, R)
24     orderByRules (R) ;
25     orderByRules (J) ;
26     time ← min (endTimes) ;
```

Firstly it is supposed that at $t = 0$ all the resources are available, so PRB tries to fit as many activities as possible in the system. Jobs that cannot start at 0 because doesn't meet requirements or because the resource are all used by other jobs, are postponed. In the Algorithm 1, the first 6 lines show the initialization of the algorithm, where with J we denote the set of activities to be scheduled, R represents the set of nodes and the function $orderByRules\,()$ order the set by a rule. The algorithm will stop when there are no jobs in the queue (line 8). Then it takes into account every job unit, checking if there are some available resources on every node by the function $checkAvailability\,(rn_i, r)$ (line 12) which takes in input a node r and the job unit rn_i and return true if there are enough available resource on node r for the job unit rn_i, otherwise false. If it is possible to fit the job unit in a node, in line 13 the algorithm updates the usage of the system with the function $updateUsage\,(rn_i, R)$, else if a job, that cannot be mapped, is registered. Line 17-21 shows that if all the job units can be mapped, it is possible to run this job and it is removed from the queue of the waiting jobs. Instead if the job cannot be mapped, the algorithm $undoUpdates\,(j, R)$ undoes the possible changes made to the system usage before with the function $undoUpdates\,(j, R)$. Finally line 24-25 computes the closest time point where resources becomes free and in line 26 the time is updated to the minimal end time of the running activities.

4.1.1 Jobs Priority Replacement for PRB

The priority rule of the PRB algorithm in this work sorts the events depending on their expected and accumulated waiting time in the queue. In particular when a job joins in the queue, a priority is assigned based on the following equation:

$$priority_j = \frac{max\,(ewt_{ji}) * qtime_j}{ewt_j} \tag{4.1}$$

The Equation 4.1, ewt_jq represents the expected waiting time of the job j for the queue q it belongs to, $qtime_j$ is the current waiting time of the job j and $max\,(ewt_{ji})$ is the maximum expected waiting time for all queues in the system. In other words jobs that are close to surpass their expected waiting time have priority over jobs that still could wait in the queue. It can happen that two or more jobs has the same priority, in this case a tiebreaker is taken into account. The tiebreaker is calculated by jobs' resource requirements multiplied by the jobs' expected duration.

$$tiebreaker_j = ewt_j * \left(j_n * \sum_{r \in res} j_r y \right) \tag{4.2}$$

In Equation 4.2, ewt_j represent the expected waiting time of the job j, j_n represent the number of requested job units, and j_r is the request of a specific resources y. Therefore is two or more jobs get the same priority, the tiebreaker of each job is calculated

and will favours jobs which require less resources and for a lower time.

We now replace the expected waiting time with the slowdown metric. Therefore we will use the slowdown metric as a criterion to give priority to the jobs waiting in the queue.

$$priority_j = \frac{T_{w,j} + T_{r,j}}{T_{r,j}} \tag{4.3}$$

In Equation 4.3 $T_{w,j}$ is the waiting time and $T_{r,j}$ is the expected duration of a job j. As a tiebreaker we will use the same criterion, showed in Equation 4.2. In order to understand better what we have changed in the pseudocode of the PRB, we now show five lines of the priority assignation:

```
1 for job ∈ queue do
2     if T_{r,j} = 0 then
3         slowdown_j = T_{w,j} ;
4     else
5         slowdown_j = (T_{w,j} + T_{r,j}) / T_{r,j} ;
```

The slowdown metric is replaced in the function *orderByRules* of the PRB algorithm. Initially, at the time $t = 0$, when all the resources are available, the slowdown is equal to 1 for those jobs which are immediately executed. At a certain time $t \neq 0$ it is possible that some resources are not free, so the slowdown is calculated for each job waiting in the queue: if the job running time is 0 the slowdown is equal to its waiting time (lines 1-3), otherwise the slowdown is calculated using the Equation 4.12.
The PRB scheduler after calculating all the jobs slowdown, gives priority to those activities which have a high slowdown value and tries to fit as many job as possible on all the nodes.

4.2 CPH Scheduler Overview

We have already discussed about the CPH in chapter 3, in this section we will look into detail this scheduler.
CPH, implemented by Borghesi et al. [19], is a combination of CP and heuristic algorithm. In this work the CP model is taken into account. CPH uses a custom search strategy guided by a branching heuristic similar to the policy of PRB.

$$\frac{max\,(ewt_{ji}) * qtime_{ji}}{ewt_j}\ \forall j \in Q \tag{4.4}$$

(4.5)

The branching heuristic gives priority to each job in the queue everytime that a new event occurs: a new job joins in the queue, or is in running or terminates its work. The criterion of this priority is described in Equation 4.4, where $max\,(ewt_{ji})$ is the maximum value of expected duration between all the jobs in the queue at the time i, $qtime_{ji}$ is the waiting time of the job j in the queue Q at the time i, and ewt_j is the expected duration of the job j. When two or more jobs have the same priority, the job demand calculated in Equation 4.2 is taken into account.

$$minz = \sum_{i=I} \frac{max\,(ewt_i)}{ewt_i}\,(s\,(\tau_i) - q_i) \tag{4.6}$$

The objective function (Equation 4.5) of the model consists in the minimization of the weighted queue time computed by the sum of the waiting times of all the jobs, weighted on estimated waiting time for each job; $s\,(\tau_i)$ is the start of the conditional interval variable and q_i is the arrival time of the job i in the system.

The model is generated everytime the scheduler is invoked, we denote this time with t. We consider every job j_i as an activity variable a_i which starts at the time $s(a_i)$, lasts $d(a_i)$ and $x(a_i) = 1$. The start time $s(a_i)$ of a job is considered as a decision variable which has as domain $[t, Eoh]$ where t is the current time and the end of the time horizon of the scheduler. The Eoh can be calculated by $min_i\,(s\,(a_i)) + \sum_i d\,(a_i)\,\forall i \in A \cup B$ where B is the set of running jobs and A is the set of waiting jobs. With a_{ikj} we refer to an activity variable created for each unit k, job i and for each possible assignment of node j. With UN_i we denote a matrix of MxP_{ij} of activity variables for every job, where M is the number of nodes in the system and Pij is the maximum number of job units dispatchable in the j the node. The schedule of CPH is generated with a relaxed model of the problem which considers each resource as a unique resource. The model is then characterized by a set of constraints, which are of the Cumulative type, one for each resource type k in the system. Then, since Eurora has four type of resources, we will have four such constraints. The model is described by four equations and by the cumulative constraint:

$$s(a_i) :: [t..Eoh] \ \forall i \in A \tag{4.7}$$

$$s(a_i) = getStart(j_i) \ \forall i \in B \tag{4.8}$$

$$x(a_{ikj}) :: [0,1] \ \forall i \in A \tag{4.9}$$

$$\sum_{j=1}^{N_n} x(a_{ikj}) = 1 \ \forall k, \forall i \in A \tag{4.10}$$

$$cumulative(a_{ikj}, req_{ikr}, cap_{jr}) \ \forall j \in N_n \forall r \in R \tag{4.11}$$

Equation 4.7 represents a set of unary constraints which limit the possible start of waiting jobs to be greater than t. The Equation 4.8 assigns the start time to the real start time already decided. Equation 4.9 assigns 0 or 1 to the allocation variables: if the job is assigned to a node, the variable is set to 1, 0 otherwise. The fourth Equation 4.10 limits the job to be assigned only to one node. The constraint in Equation 4.11 is the cumulative constraint, where the variable req_{ikr} is the amount of resources required by each job unit k of job i for a resource $r \in R$ in the set of resources Res and cap_{jr} is the variable representing the capacity for each resource $r \in Res$. The cumulative constraint limits the resource usage to stay below the resource capacity for each resource type at any time. Regarding the solution, it is not necessary to find an optimal solution but we use a timelimit criterion to bound the search. When the timelimit expires, the best solution found is the scheduling decision.

4.2.1 Jobs Priority Replacement for CPH

We now how we replaced the expected waiting time with the slowdown metric in the CPH scheduler. As previously mentioned, CPH uses the expected waiting time as a criterion to give priority to the jobs in the queue in the search (Equation 4.4) and in the objective function (Equation 4.6).

The new objective function is showed in Equation 4.12. The latter consists in a minimization of the sum of all the jobs slowdown in the queue. Basically we are giving priority to jobs with high slowdown.

$$min \ z = \sum_{i=I} \frac{T_{w,j} + T_{r,j}}{T_{r,j}} \tag{4.12}$$

The heuristic search of CPH is introduced in Equation 4.4. As we can see, the CPH search uses a similar criterion to the PRB jobs priority i.e. they are giving priority to jobs through expected waiting time. We now replace this criterion with the slowdown metric in a similar way of PRB scheduler. Therefore we use the Equation 4.3 instead of

Equation 4.4 in order to give priority to the jobs when the CPH search is invoked.

Overall we have four different CPH variants, depending on the jobs priority assignation:

- we let *CPH-0* denote the original CPH which uses the expected waiting time as a priority both in the objective function and in the heuristic search.
- we let *CPH-1* denote the CPH which uses as jobs priority the slowdown in the heuristic search and the expected waiting in the objective function.
- we let *CPH-2* denote the CPH which uses the expected waiting time to give jobs priority in the heuristic search and the slowdown metric in the objective function.
- we let *CPH-3* denote the CPH which uses the slowdown metric as a priority criterion both in objective function and in the heuristic search.

4.3 Pure CP Scheduler Overview

We give now and overview of the pure CP, mentioned in the previous chapter, which denote the CP scheduler implemented by Bridi et al. [1]

The model is of the pure CP scheduler is similar to the CPH and is composed by a set of cumulative constraints, one for each resource type, and by conditional interval variables [21]. The main differences from CPH is that pure CP uses the alternative constraint:

$$alternative\left(a_i, UN_{ikj}, k_i\right) \ \forall i \in A \tag{4.13}$$

The constraint in Equation 4.13 gives the possibility for every job unit to be displaced in a node partially used by another job unit of the same job.

$$minz = \sum_{i=1}^{n} \frac{s\left(a_i\right)) - stq_i}{ewt_{queue(j_i)}} \tag{4.14}$$

The objective function, showed in Equation 4.14, of the model consists in the minimization of the sum of all the waiting times of each job weighted by the expected waiting time of the queue where the job is submitted.

The original search of the pure CP is the default of the IBM ILOG which is a C++

library for solving complex combinatorial problems in diverse areas including production planning, resource allocation, time tabling, personnel scheduling, cutting materials, blending mixtures, assigning radio frequencies, and many others. The algorithm adopted by the ILOG solver is the "Self-Adapting Large Neighborhood Search", since we are testing the schedulers with python as a programming language and OR-Tools as a constraint solver, we have adopted the default search of this solver. Or-Tools uses the "Large Neighborhood Search" and is composed in three main steps:

1. It starts with a solution which has to be feasible: if there are not feasible solution, constraints are relaxed until is possible to construct a starting solution for that relaxed model. From here, the relaxed constraints are enforced by adding corresponding terms in the objective function.

2. improves locally this solution: defines a neighborhood for a given solution and a way to explore this neighborhood, frequently closed solutions are used (i.e. they belong to the same neighborhood). The search explores a neighborhood around an initial solution, if it find a better solution, it takes it and updates it with the latter, and restarts all over again until a stopping criterion is met. If we denote with N_x the neighborhood of a solution x, the local search of OR-Tools is described as follows:

Algorithm 2: OR-Tools Local Search

```
1 x ← x_0 ;
2 while stop criteria not met do
3     Find neighborhood N_x ;
4     Find the best solution in N_x : x_best ;
5     x ← x_best ;
```

3. The search finishes when reaching criteria: is possible use a time limit for the whole solving process or for some parts of the solving process. We can limit the search using a maximum number of steps/iterations,branches,failures,solutions. In the pure CP, like CPH, a timelimit is used to solve the whole search process.

4.3.1 Jobs Priority Replacement for pure CP

For the pure CP scheduler we have replaced ,using the similar method of the CPH described in Section 4.2.1, the expected waiting time with the slowdown metric in Objective function and in the search.

Regarding the objective function, we have replaced the job priority criterion based on expected waiting time, previously showed in Equation 4.14, with the job priority criterion based on slowdown similar to Equation 4.12

We have modified the search of the pure CP, which was not using a priority criterion, with an heuristic search that gives priority to jobs with higher slowdown. The heuristic search is basically the same adopted by the CPH and showed in Equation 4.3.

Therefore pure CP, like CPH, has four different jobs priority variants:

- we denote with *CP-0* the base version of the pure CP implemented by Bridi et al. [1] which uses as objective function, the minimization of the sum of all the job waiting time weighted by the expected waiting of the queue where the job is submitted.
- we denote with *CP-1* the variant in which the slowdown metric is used as a priority criterion in the branching heuristic.
- we denote with *CP-2* the variant in which the expected waiting time priority criterion is replaced in the objective function.
- we denote with *CP-3* the variant which uses the priority based on the slowdown in both objective function and in the search.

We will show in the following chapter the performances achieved by the two constraint programming schedulers and by the PRB scheduler.

Chapter 5

Experimental Results

In this chapter we will look at the experimental results obtained with the schedulers showed in the previous chapter, tested with different priorities criteria, on the Eurora workload. We will then draw our conclusions, after having described the performances of each scheduler.

In Section 5.1 we will describe the test methodology, together with the evaluation metrics that will be used. In Section 5.2 we will then look at the results obtained by each scheduler and their different priorities. In Section 5.4, we will draw our conclusions about the utilization of the slowdown metric in job scheduling priority over the data-driver approach.

5.1 Methodology

We used for the experiments the three variants of the Eurora workload which, as we mentioned previously, correspond to the wall time (W), real (R) and the data driven duration (D) of the jobs. Each of these three variants is tested with all the schedulers and their priorities variants previously introduces for a total of 42 tests. Since we run test on the Eurora workload, which include 372320 jobs, 7 million jobs have been simulated and analyzed except for the pure CP which is tested over two months, for a total of 76265 jobs due its scalability.

The used metrics are the slowdown, the queue size and the waiting time. For a better visualization we plot only jobs where slowdown is different from 1. We remove those jobs because are immediately scheduled as they arrive in the system, for that reason they are not relevant for our comparison. The waiting time is measured in seconds: in our tests we do not take in consideration jobs with waiting time equal to 0 but we consider

them in the calculation of the average. While the queue size is measured by the number of jobs waiting at every seconds.

The tests were performed on a desktop computer equipped with Intel I5 6400 (with a processor base frequency of 2,70 GHz), 8 GB and, as OS, Windows 7. The schedulers that took the most time were pure CP and CPH, since they are constraint programming based schedulers and sophisticated. In general CPH took around 12-20 hours to schedule each variant of the entire workload, PRB took around 6-12 hours and the pure CP took around 24 hours to schedule 76265 jobs.

All the schedulers are written in Python, we used a specific library to test the constraint programming schedulers. The library, already introduced in the previous chapter, is OR-Tools; it implements various models and search algorithms for constraint programming and optimization in general.

The representation of the results is shown through boxplot: a graphical representation used to describe the distribution of a sample through simple dispersion and position indices. The minimum an maximum values of the experiments are represented with a top and a bottom horizontal line, the coloured box represent the difference between the 3rd and 1st quartiles, that is, the width of the range of results that contains the "central" half of the observed results. The median is represented by an horizontal line within the box and the mean by a green triangle. The waiting times plot, besides boxplot, uses a trend plot, showing the trend of the average waiting time, calculated over all the jobs and including those jobs which does not wait, over all the considered schedulers.

5.1.1 The Accasim simulator

All the schedulers have been tested on **Accasim**, a simulator for HPC systems. Accasim is an open-source HPC system simulator focused on the evaluation of dispatching methods. It was implemented for represent various real HPC system resources, developing dispatching methods and to carry experiments across different workload sources. Using a real HPC system for experiment is not possible because researches may not have access to a real system and because it is impossible to modify the hardware components, while with Accasim is highly customizable and we can define resources settings, workload sources and dispatching methods. In order to let Accasim work, the user must implement a parser for the workloads format, in order to correctly read and use it. By default, AccaSim supplies a parser for the **Standard Workload Format** (SWF).

The most important parts of the simulator are the *dispatcher* that decides which jobs waiting in the queue to run next (*scheduler*) and on which resources to run them (*allocator*) maintaining high system performance and high utilization of resource and the *resource manager* which update the updates the system status through a set of active monitors. The simulator generates in output three usefull files:

- the *scheduling file* contains records about dispatched jobs in the workload;
- the *pretty print file* is similar to the scheduling file but it is more "humanly readable";
- the *statistics file* contains statistic informations of the simulation;

We have configured Accasim to have the same resources of the HPC system Eurora and to work with all the schedulers previously introduced. The simulator takes as input the Eurora's workload in order to have a similar behaviour of the real HPC. We took the output files in order to understand the performances of each scheduler and analyze their different jobs priority.

5.2 Tests Results of the PRB Scheduler

In this section we will present the results obtained by the simulation of the PRB scheduler on the entire workload of Eurora. We will analyze these results using three different metrics: the slowdown, the queue size and the waiting time. Moreover we denote with SDPRB (slowdown PRB) the scheduler which uses the slowdown measure to give priority to the queued jobs.

5.2.1 Slowdown Analysis

Figure 5.1 shows the distribution of slowdown on the entire workload simulated with the normal PRB and on its slowdown variant, the SDPRB. As we can see, SDPRB performs better than PRB. The original scheduler, in terms of efficiency, has same results over the three duration types, but the worst performance is achieved by the simulation of PRB over the data-driven duration time. Moreover, SDPRB along with the real job duration reaches the highest performance in terms of slowdown among the three different types of job durations. However we use it for understanding if our prediction is close to the real duration since it is not possible to know it a priori. The PRB with the data-driven prediction is improved by the slowdown metric priority, indeed achieves a better result than SDPRB-W, PRB-W,PRB-D and PRB-R. The SDPRB with walltime declared by

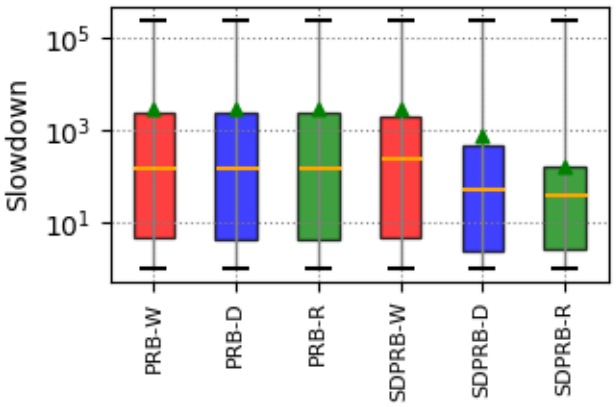

Figure 5.1: Distribution of job slowdown for PRB methods

the user is the worst and achieves a similar result as the PRB method with all the three duration types.

In Figure 5.2, the slowdown distribution for short jobs, medium jobs and long jobs is depicted. For short jobs, as well as in Figure **??**, SDPRB is better than PRB, and the data-driver prediction has improved with the new variant of PRB. This distribution is similar the one previously shown, due to the fact that short jobs has a frequency of 93.15 % in the workload. If we observe the medium jobs distribution, we have low values of slowdown, which means that medium jobs does not wait too much before being scheduled, but even in this case, SDPRB overcome PRB. SDPRB with our estimated waiting time is better than PRB over all job duration types, except for the SDPRB with real duration. Even in this slowdown distribution, the data-driven prediction with the new PRB is improved. The scale metric of long jobs in this graph is not logarithmic but linear because of low values of slowdown. The normal PRB dispatcher achieves better results than the SDPRB, because we think that SDPRB underestimates the duration of long jobs and because the slowdown of long jobs increases much more slowly than short jobs and medium jobs. The SDPRB with the walltime declared by the user has almost the same median of SDPRB-D, which means that most of the jobs are concentrated around the orange line but the mean is higher than SDPRB-D. In this graph PRB with all job duration types is better than SDPRB.

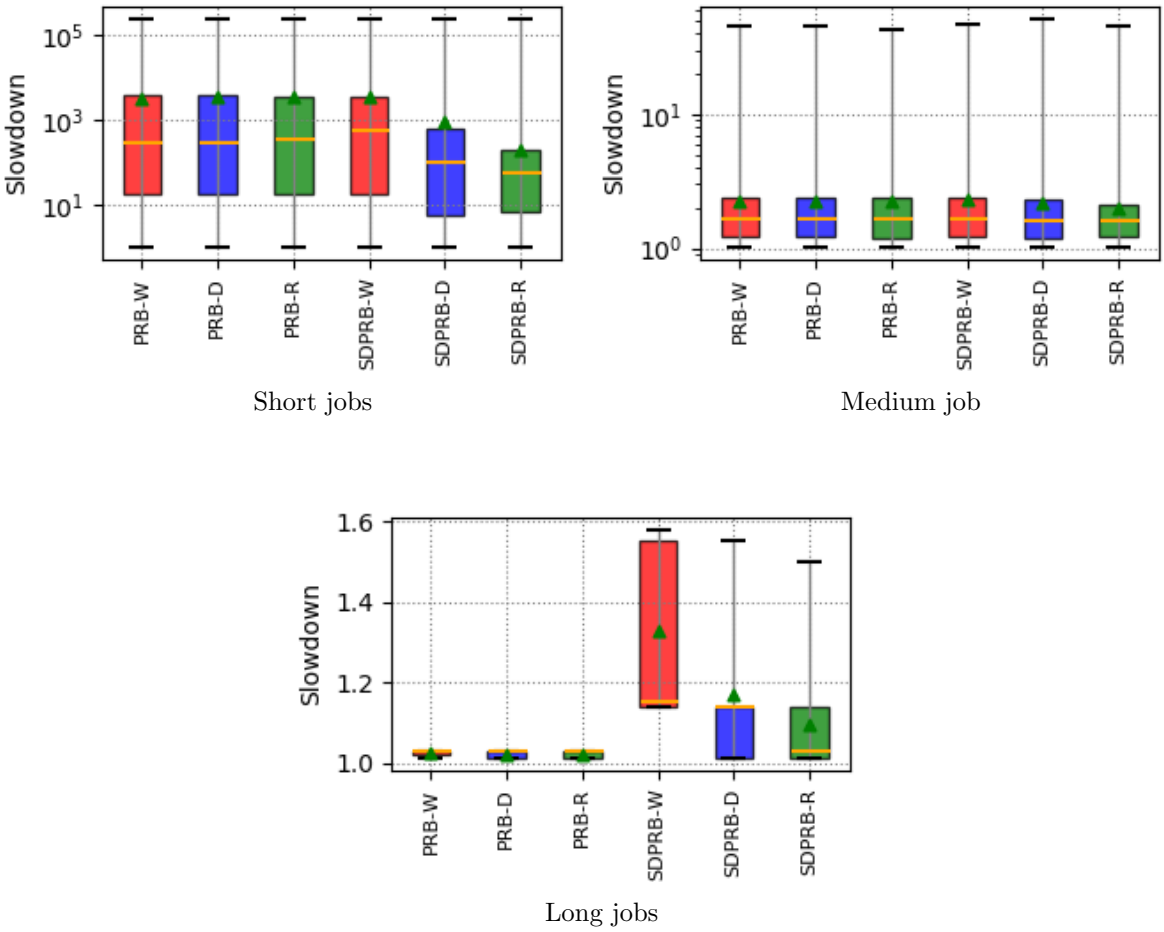

Figure 5.2: Distribution of job slowdown for short, medium and long jobs

5.2.2 Queue Size Analysis

Figure 5.3 shows the distribution of the number of jobs in the queue at every seconds for PRB and its slowdown variants. We can see that the effect on the system is similar to the performance measured by the slowdown. PRB shows no differences over all the duration types, while SDPRB with the data-driven approach achieves a similar results achieved by the performances of SDPRB over the real duration. The slowdown metric in the priority assignation has improved the PRB performances over the data-driven workload in terms of queue size. The best result, in terms of average and median is obtained by the SDPRB over the real workload.

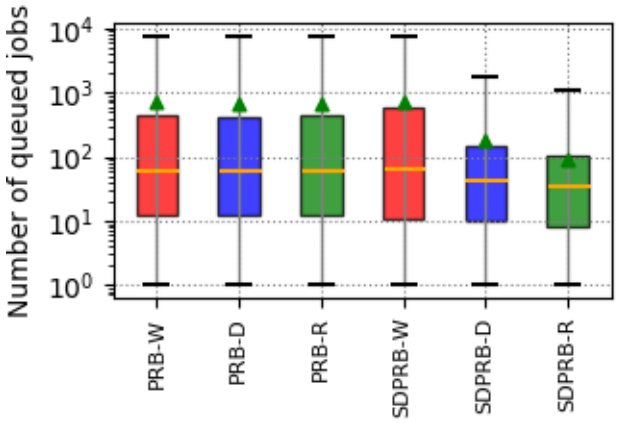

Figure 5.3: Distribution of number of jobs waiting at every seconds, for each PRB method

5.2.3 Waiting Time Analysis

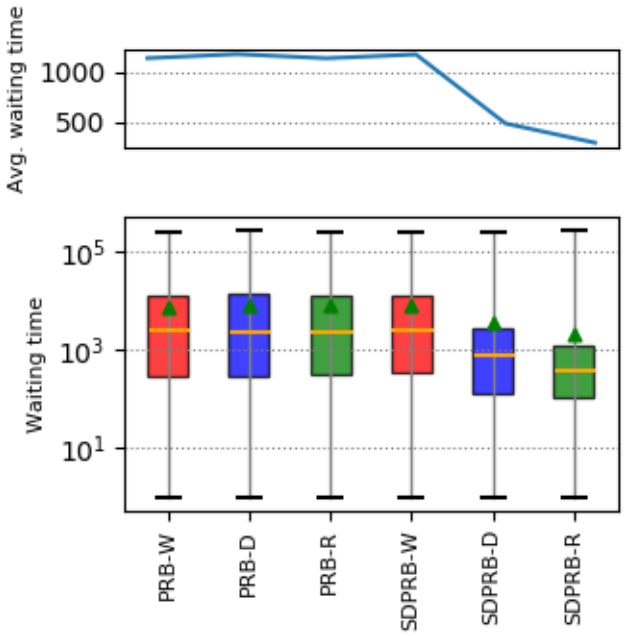

Figure 5.4: Distribution of the jobs' waiting times for each PRB method

In Figure 5.4 the distribution of the jobs' waiting times for the PRB is depicted. The normal PRB shows a sort of balance among the three jobs' duration types with almost the same average waiting time. A similar result is achieved by SDPRB over the walltime

job duration. We can see a good improvement when we use the slowdown metric on the data-driven job duration. Indeed, the performances of the latter are improved compared to PRB-D and PRB-R. However the best performance obtained always remains at the SDPRB over the real jobs duration.

5.3 Tests Results of the CPH Scheduler

This section presents the results obtained by the simulation of the CPH scheduler with different priority, showed in Section 4.2.1 on the entire workload of Eurora. We will analyze, in a similar way of the previous section, the results using three different metrics: the slowdown, the queue size and the waiting time.

5.3.1 Slowdown Analysis

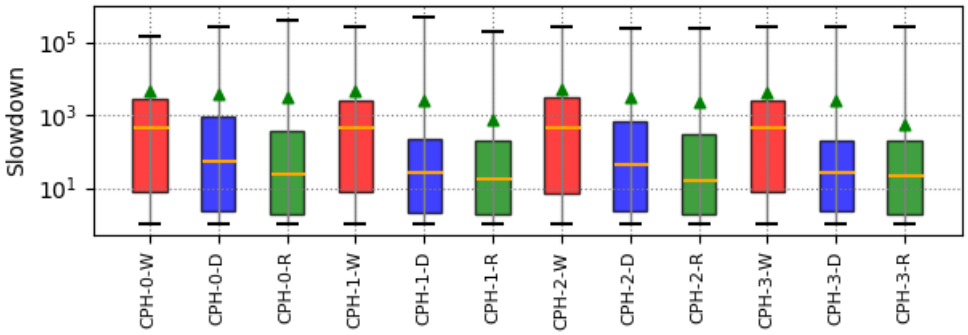

Figure 5.5: Distribution of job slowdown for each CPH method

Figure 5.5 shows the distribution of job slowdown on the entire workload, simulated with the all the CPH priority methods. As we can see, the slowdown distribution of the walltime workload is similar over all CPH variants, this means that the priority based on the slowdown metric does not always benefits the CPH methods using the walltime job duration. If we look at the data driven results, the best performance is achieved by CPH3 (variant with the slowdown metric replaced in the branching heuristic priority and in the objective function priority), however all the CPH variants show an improvement compared to the normal CPH-0, which uses the expected waiting time in both branching heuristic and objective function. The best results obtained by the schedulers in this picture is the CPH1-D compared to CPH1-R and CPH3-D compared to CPH3-R, where in both cases CPH-1 and CPH-3 on the data-driven workload achieved similar results to those of the real workload.

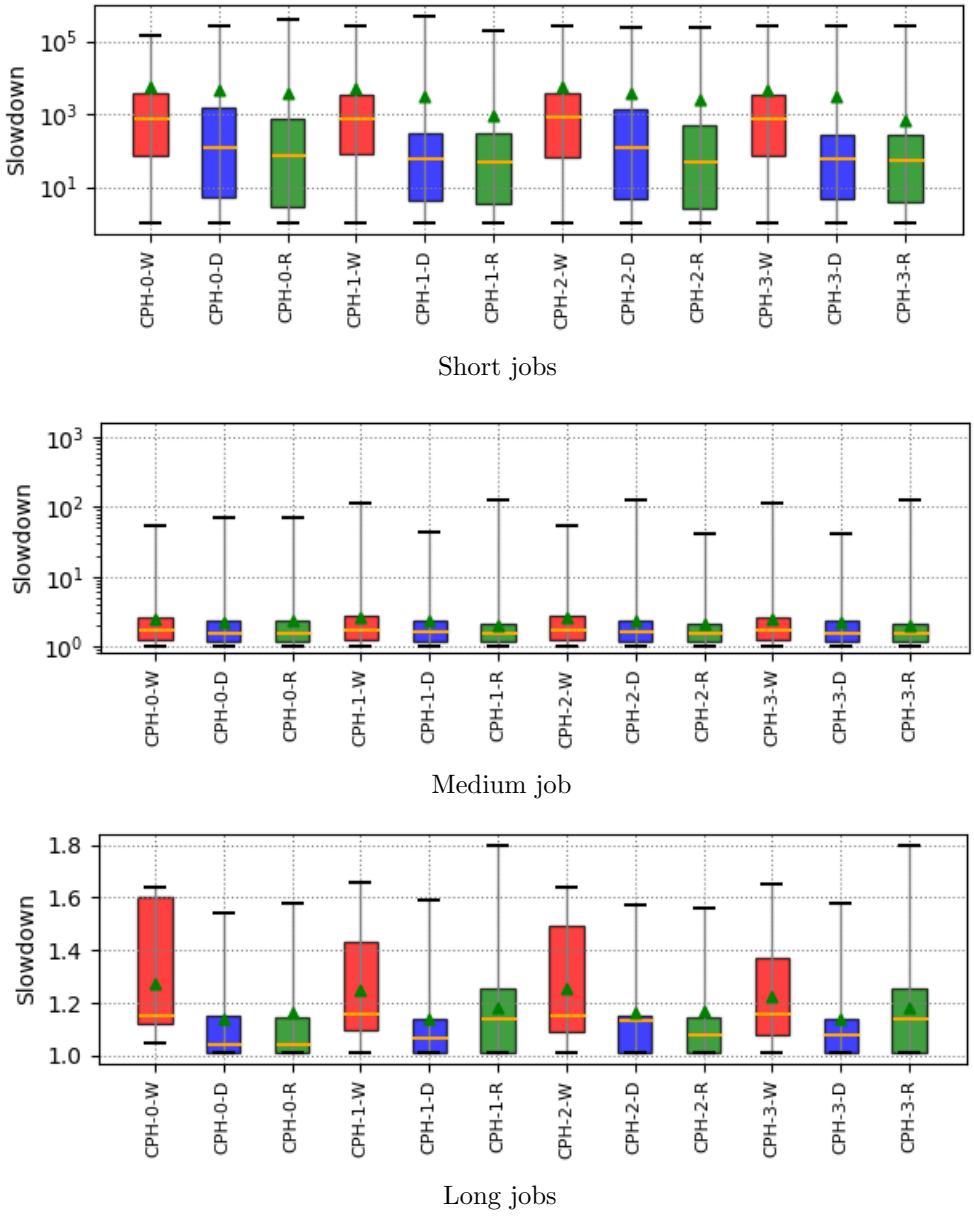

Figure 5.6: Distribution of job slowdown for short,medium, and long jobs

In Figure 5.6 the slowdown distribution for short medium, and long jobs. If we look at the short jobs distribution, the results are similar to the previous distribution. Here we can see better how similar are CPH-1 and CPH-3 when using the data-driver workload and real workload. The boxplots have almost the same dimension and the medians as well. The distribution of the walltime duration over CPH-0 achieved the same results

of CPH variants over the walltime workload. In the medium jobs distribution, we have low slowdown values which means that medium jobs do not wait too much before being schedule. We can see that CPH 1 with real job duration performs better than the other variants. All the distribution of the data-driven job durations are similar, therefore, in this case, job slowdown does not improve the scheduling of medium jobs. We can see the same behaviour on the walltime duration. In the distribution of job slowdown for long jobs, the walltime job duration over all the methods achieved the worst performance especially with CPH-0. An interesting result is obtained by CPH-3-D and by CPH-1-D which are better than their real job duration while with others variants both real and data-driver duration get same performances.

5.3.2 Queue Size Analysis

Figure 5.7 depicts the queuesize distribution for CPH and its priority variants. The results obtain by the CPH variants, compared to the base version, are improved. The best performance in this plot is achieved by CPH-3-D, getting almost a perfect prediction to the real duration. Overall, the queue size results confirmed what slowdown results showed before, in fact the queue size is a parameter that depends primarily on the scheduler, which has full control over the job queue. The slowdown metric in the CPH scheduler has improved the performances of the system with the data-driven workload. The only one which did not get any benefit is CPH-2 (which uses the slowdown metric priority in the objective function), and its performances over the three job duration types are comparable with those achieved by the base version, CPH-0.

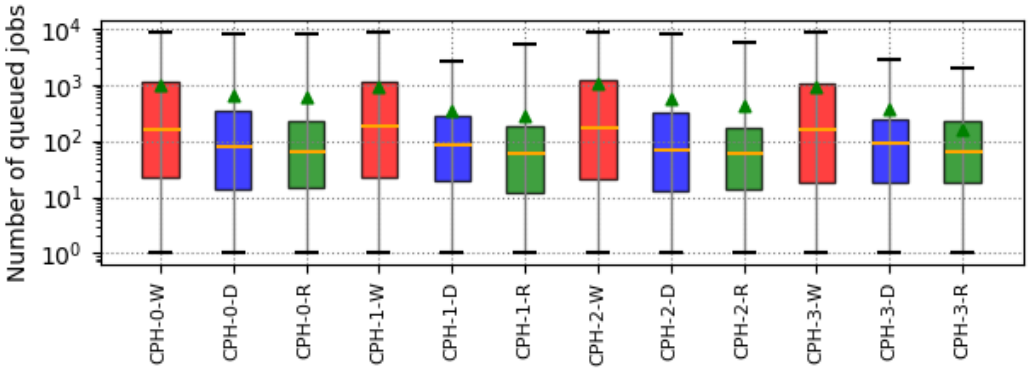

Figure 5.7: Distribution of number of jobs waiting at every seconds, for each CPH method

5.3.3 Waiting Time Analysis

Figure 5.8 shows the distribution of the jobs' waiting times generated by the simulation of all the CPH variants over the three workloads. The obtained results are quite similar to the performances previously showed with the slowdown metric and with the queue size metric. The data driven job duration has improved in all the variants. If we look in the plot of the average waiting time, we can see better that the trend of the line is more higher in the CPH-0 zone, while after the latter, the trend achieves better values. In general, the slowdown metric used as priority criteria has brought benefits, in terms of waiting time, on the CPH over the data-driven job duration.

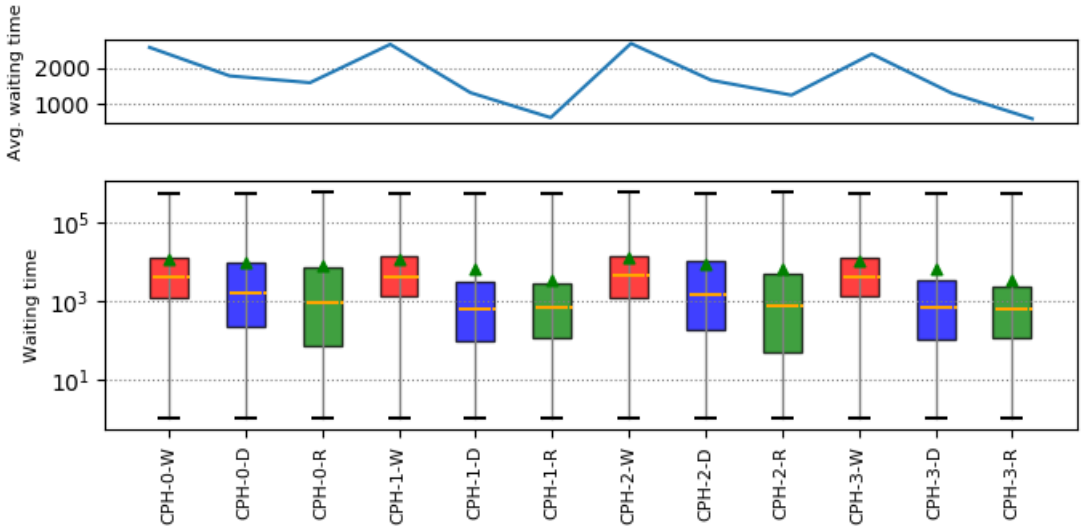

Figure 5.8: Distribution of the jobs' waiting times for each CPH method

5.4 Tests Results of the pure CP Scheduler

This section shows the performances achieve by the simulation of the pure CP scheduler with different priority, showed in Section 4.3.1. Due to its low scalability, we have tested the scheduler with a restricted workload, where the number of jobs amounts to 76265. We will analyze, in a similar way of the previous section, the results using three different metrics: the slowdown, the queue size and the waiting time.

5.4.1 Slowdown Analysis

In Figure 5.9 the performance in terms of slowdown is shown for each priority variant of the pure CP. We can see a small improvement of the performances of pure CP over the data-driven job duration. In particular CP-1 and CP-2 achieved a good performance, better than the one obtained by CP-0 and CP-3. We can see that the slowdown metric, used as a criterion of jobs' priority, in the objective function, for the first time has achieved a result better than the one achieved the CP base version, which uses the expected waiting time in the jobs priority. The best performance, in terms of data-driver jobs' duration, is obtained by CP-1 and we can see that the difference between the latter and the CP-1-R is small. The worst results are obtained by CP-2-W which has the highest slowdown average.

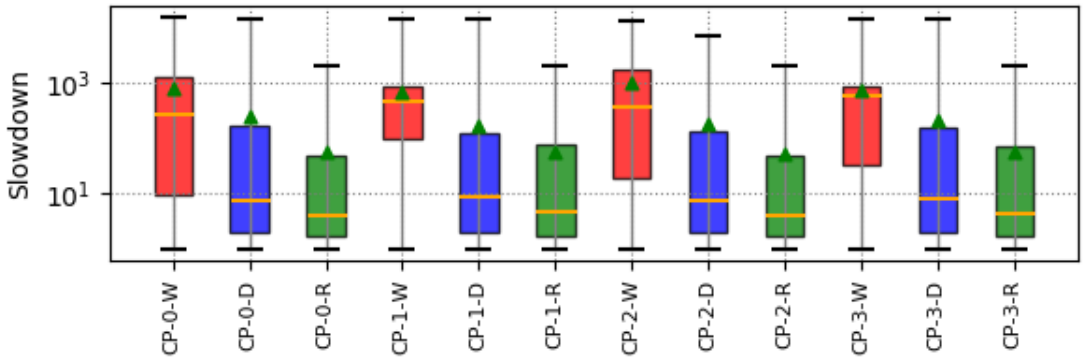

Figure 5.9: Distribution of job slowdown for each pure CP method

5.4.2 Queue Size Analysis

Figure 5.10 depicts the queue size distribution for the pure CP and its priority variants. A small improvement is visible over the data-driven workload obtained by CP-3, in fact its average is lower than the average of the other priority variants. The queue size distribution of all the methods over the real job duration is similar. The worst performance is achieved by CP-0, the base version, over the walltime workload.

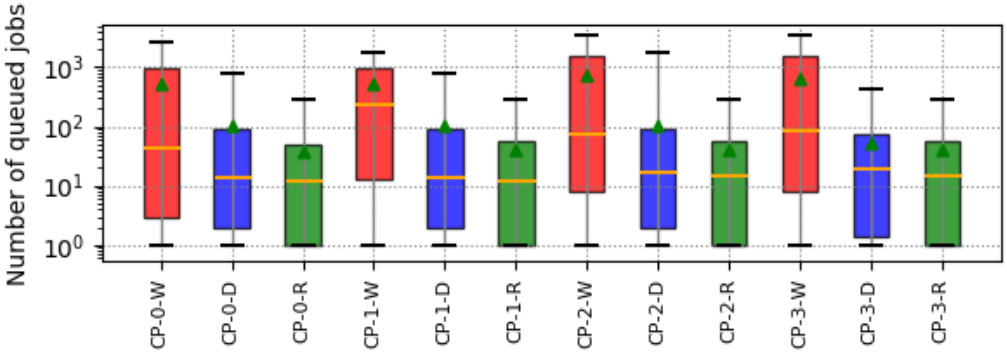

Figure 5.10: Distribution of number of jobs waiting at every seconds, for each pure CP method

5.4.3 Waiting Time Analysis

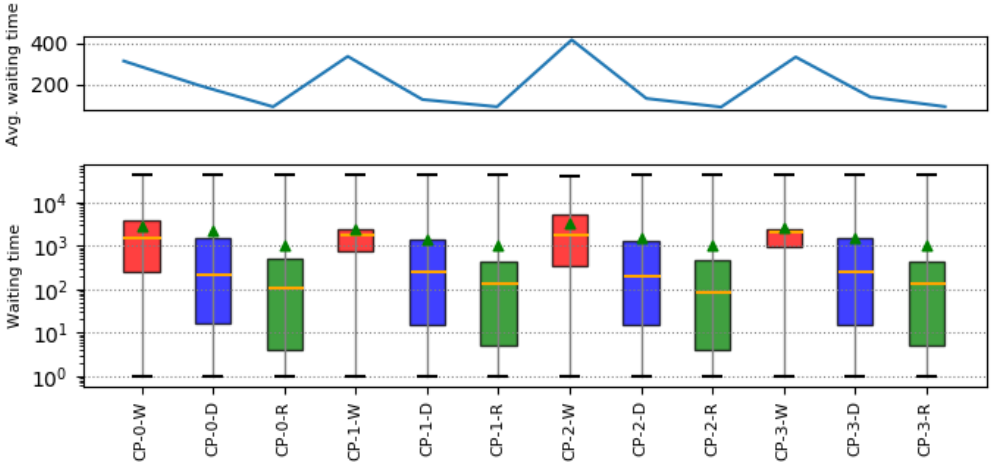

Figure 5.11: Distribution of the jobs' waiting times for each pure CP method

Figure 5.11 shows the waiting time distribution of all the pure CP priority variants. We can see a small improvement of the methods which use the slowdown metric as a priority, in the plot of the average waiting time over the data-driven workload. The performance achieved by each method are almost all similar to their variants except CP-2-W which has the most higher average and median. Overall all the CP priority variants achieved a similar results on the real jobs' duration and on the data-driven jobs' duration while the walltime remained the worst.

5.5 Experimental Observations

The PRB scheduler has improved the performances of the base version, which uses the expected waiting time as a criterion of jobs priority, with the slowdown metric in the priority rule. In fact, as we can see in Section 5.2, the performances achieved over the data-driven job duration are almost always better than the base version, in terms of slowdown, queue size and waiting time.

The results obtained by the CPH showed, in terms of slowdown, queue size and waiting time, has led to an improvement on the data-driven job duration. CPH-1 has not showed a particular benefit, in fact it has achieved similar performances to the CPH which uses the expected waiting time to give jobs priorities. CPH-1 and CPH-3 instead, achieved the best performance, similar to those obtained by their simulation over the real job duration.

Pure CP instead, has not showed a relevant improvement, we think that the cause of the latter may be because it has been tested with a restricted workload due to its low scalability. However we observed a small improvement on the data-driven job duration, in terms of slowdown, is achieved by CP-1, which uses the slowdown metric as a priority criterion in the objective function.

We have observed that the slowdown metric priority has improved primarily the short jobs, while longs and medium are not affected by this improvement. The cause of the latter may be because the slowdown metric over-emphasizes the importance of very short jobs [20]. For example, if we consider a job with 60 seconds of duration that is delayed for 10 minutes has a slowdown of 11, whereas a job of 3600 seconds delayed by the same 10 minutes has a slowdown of only 1.16 however the Eurora workload is made of 372320 jobs in which 93.15% fall into the the short class and the choice to use the slowdown metric as a job priority, has led to great benefits, especially to the data-driver job duration.

Chapter 6

Conclusions and Future Work

The present section shall provide an overview of the research that was conducted in this thesis and, eventually, provide some suggestions for future work that is possible in this direction. Therefore, here is a recap of the assumptions and their applications that have been taken into account: in the hereby presented work we presented an analysis of the effect of job duration prediction on HPC job scheduling. We considered a real workload dataset collected by Eurora, a hybrid HPC system. We have used three different schedulers, two of which are based on constraint programming, and changed their job priority according to be the slowdown metric. As a further step, we have studied their performances in the presence of prediction based on a data-driven approach, on estimates based on wall-time, using as a baseline the real job duration. In the end, we have compared their performance after and before their job priority replacement in order to investigate if there have been improvements comparing to the original performances on the data-driven prediction.

Turning to illustrate the outcomes of the described research, we shall underline that thanks to the analysis of the results obtained by each scheduler we have understood that the slowdown metric, used as a priorities' assignment criterion for the jobs in the queue has improved the performances of schedulers, taken into account on the data-driven job duration.

Future work will presumably regard the application of the slowdown metric as a priority criterion in other HPC schedulers in order to investigate how they can benefit from the data-driven prediction. Another suitable option for future work could be a research on testing the modified schedulers showed in this thesis in different HPC workloads. Additionally, we plan to test the pure CP on the whole EURORA workload in order to understand the effects of the new job priority with this scheduler.

Chapter 7

Acknowledgments

My sincere acknowledgements go to my supervisor professor Zeynep Kiziltan, for giving me the opportunity to work on this thesis. I would also like to thank Cristian Galleguillos who helped me for this work.

I want to thank all my friends from my hometown, that have always supported me, and that everytime make me laugh and have fun. I would like to thank all of the friends that I have made here in Bologna, it is always nice to be with you my friends.

A special acknowledgements goes to Cristina, that has always been by my side and that has helped and supported me in these years. I shared fantastic memories here in Bologna and you took me to the position I am currently now.

My best acknowledgements go to my family, which allowed me to study away and that were always there for me.

Bibliography

[1] T. Bridi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini. A constraint programming scheduler for heterogeneous high-performance computing machines. IEEE Transactions on Parallel and Distributed Systems, 27(10):27812794, October 2016.

[2] D. H. Ahn, J. E. Garlick, M. A. Grondona, D. A. Lipari, R. R. Springmeyer. A High Performance Computing Scheduling and Resource Management Primer. *Lawrence Livermore National Laboratory*, LLNL-TR-652476, March 2014.

[3] F. Rossi, P. v. Beek, T. Walsh. Handbook of Constraint Programming (Foundations of Artificial Intelligence). Elsevier Science Inc., New York, NY, USA, 2006.

[4] Hu, X., Eberhart, R. C., Shi, Y. (2003, April). Swarm intelligence for permutation optimization: a case study of n-queens problem. In Swarm intelligence symposium, 2003. SIS'03. Proceedings of the 2003 IEEE (pp. 243-246). IEEE.

[5] H. Fu, J. Liao, J. Yang, L. Wang, Z. Song, X. Huang, C. Yang, W. Xue, F. Liu, F. Qiao, et al. The Sunway TaihuLight supercomputer: system and applications. In: Science China Information Sciences 59.7 (2016), pp. 116.

[6] Hovestadt, M., Kao, O., Keller, A., & Streit, A. (2003, June). Scheduling in HPC resource management systems: Queuing vs. planning. In Workshop on Job Scheduling Strategies for Parallel Processing (pp. 1-20). Springer, Berlin, Heidelberg.

[7] Dhar, V. (2013). Data science and prediction. Communications of the ACM, 56(12), 64-73.

[8] Zhang, Y., Squillante, M. S., Sivasubramaniam, A., & Sahoo, R. K. (2004, June). Performance implications of failures in large-scale cluster scheduling. In Workshop on Job Scheduling Strategies for Parallel Processing (pp. 233-252). Springer, Berlin, Heidelberg.

[9] Chen, X., Lu, C. D., & Pattabiraman, K. (2013, June). Predicting job completion times using system logs in supercomputing clusters. In Dependable Systems and

Networks Workshop (DSN-W), 2013 43rd Annual IEEE/IFIP Conference on (pp. 1-8). IEEE.

[10] Matsunaga, A., & Fortes, J. A. (2010, May). On the use of machine learning to predict the time and resources consumed by applications. In Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (pp. 495-504). IEEE Computer Society.

[11] Srbu, A., & Babaoglu, O. (2016, August). Power consumption modeling and prediction in a hybrid CPU-GPU-MIC supercomputer. In European Conference on Parallel Processing (pp. 117-130). Springer, Cham.

[12] C. Galleguillos, Z. Kiziltan, and A. Netti. AccaSim: an HPC Simulator for Workload Management. In: To appear in High-Performance Applications and Tools in Latin America High-Performance Computing Conference. 2017.

[13] C. Galleguillos, A. Sirbu, Z. Kiziltan, O. Babaoglu, A. Borghesi, and T. Bridi. Data-driven job dispatching in HPC systems. In: To appear in Proceedings of the Third International Conference on Machine Learning, Optimization and Big Data (MOD 2017). 2017.

[14] Mersmann, O., Bischl, B., Bossek, J., Trautmann, H., Wagner, M., & Neumann, F. (2012). Local search and the traveling salesman problem: A feature-based characterization of problem hardness. In Learning and Intelligent Optimization (pp. 115-129). Springer, Berlin, Heidelberg.

[15] Baptiste, P., Laborie, P., Le Pape, C., & Nuijten, W. (2006). Constraint-based scheduling and planning. In Foundations of Artificial Intelligence (Vol. 2, pp. 761-799). Elsevier.

[16] Laborie, P., & Rogerie, J. (2008, May). Reasoning with Conditional Time-Intervals. In FLAIRS conference (pp. 555-560).

[17] Baraglia, R., Capannini, G., Pasquali, M., Puppin, D., Ricci, L., & Techiouba, A. D. (2008). Backfilling strategies for scheduling streams of jobs on computational farms. In Making Grids Work (pp. 103-115). Springer, Boston, MA.

[18] Bartolini, A., Borghesi, A., Bridi, T., Lombardi, M., & Milano, M. (2014, September). Proactive workload dispatching on the EURORA supercomputer. In International Conference on Principles and Practice of Constraint Programming (pp. 765-780). Springer, Cham.

[19] Borghesi, A., Collina, F., Lombardi, M., Milano, M., & Benini, L. (2015, August). Power capping in high performance computing systems. In International Conference on Principles and Practice of Constraint Programming (pp. 524-540). Springer, Cham.

[20] Feitelson, D. G. (2001, June). Metrics for parallel job scheduling and their convergence. In Workshop on Job Scheduling Strategies for Parallel Processing (pp. 188-205). Springer, Berlin, Heidelberg.

[21] Laborie, P., & Rogerie, J. (2008, May). Reasoning with Conditional Time-Intervals. In FLAIRS conference (pp. 555-560).