# Model Design and Accuracy for Resource Management in HPC

Guillaume Pallez

**- UNIVERSITÉ DE BORDEAUX -**
Laboratoire de Recherche en Informatique - UMR5800 - LaBRI

*en vue d'obtenir le diplôme*

**Habilitation à Diriger des Recherches,**

*École Doctorale de Mathématiques et d'Informatique*

*présentée et soutenue publiquement le 11/07/2023 par*

Guillaume PALLEZ

# Model Design and Accuracy for Resource Management in HPC

*Devant la commission d'examen formée de :*

| | | |
|---|---|---|
| Florina CIORBA | University of Basel | *Rapporteuse* |
| Johane COHEN | CNRS | *Examinatrice* |
| Zhiling LAN | Illinois Institute of Technology & Argonne National Laboratory | *Rapporteuse* |
| Arnaud LEGRAND | CNRS | *Rapporteur* |
| Krzystof RZADCA | Google & University of Warsaw | *Examinateur* |
| Samuel THIBAULT | University of Bordeaux | *Président du jury* |
| Emmanuel JEANNOT | Inria | *Garant* |

*"All models are wrong, but some are useful."*

– George Box, 1976

# Remerciements

Je voudrais d'abord remercier mes rapporteurs et rapportrices: Florina Ciorba, qui a accepté de remplacer au pied levé un rapporteur qui a du annuler pour des raisons personnelles, Zhiling Lan, qui par ailleurs subit une soutenance en visio à 7h du matin, et Arnaud Legrand, qui, comme c'est bien connu des chercheurs Grenoblois, a probablement du annuler quelques séances de ski ou de tennis pour écrire son rapport.

Je suis également reconnaissant aux membres du jury, en particulier à Krzysztof Rzadca qui interrompt ses vacances en France pour venir à cette HDR, ainsi qu'à Johanne Cohen et Samuel Thibault qui eux mêmes auraient peut être préféré être sur la plage ce 11 juillet. Enfin je remercie Emmanuel Jeannot pour sa relecture du manuscrit, ainsi que d'avoir accepté le rôle de "garant" (dont nous ne sommes toujours pas trop sûr de ce que ça veut dire exactement).

Une dizaine d'année depuis ma thèse, ça fait beaucoup de choses, beaucoup de rencontres, en France comme à l'étranger (principalement aux États-Unis d'ailleurs).

Je voudrais remercier l'ensemble des gens avec qui j'ai pu interragir scientifiquement, que ce soient mes co-auteurs et co-autrices, mais aussi celles et ceux avec qui j'ai discuté de science sans forcément publier ensemble. Des collègues avec qui je travaille depuis 15 ans (et qui m'ont même relu l'HDR), aux thésards avec qui je commence à travailler, en passant par certains collègues du SED[1]. Ceci est possible grâce aux différentes rencontres et discussions aléatoires lors de divers événements scientifiques, autour du babyfoot, ou même en se retrouvant dans un bureau commun par manque de place dans les locaux. La science est collaborative et ce sont vraiment toutes ces interractions qui font que ma recherche est ce qu'elle est aujourd'hui.

En relisant mon HDR, je me rencontre d'ailleurs que celle-ci a également été influencée par l'écriture du cours de M2 "Algorithms for High-Performance Computing Platforms" (donné avec Francieli puis avec Laercio) à l'Enseirb Matmeca, et en ça je remercie les responsables du master de nous avoir fait confiance. Ce cours m'a aidé à prendre du recul sur les difficultés autour des problèmes de modélisation et de définition des problèmes qui sont au cœur de ce manuscrit.

Je voudrais ensuite remercier toutes les personnes sur lesquelles j'ai reposé pendant l'écriture de cette HDR (qui j'espère se reconnaitront), que ce soit dans le cadre professionnel, mais également personnel (merci à la *Table Basse* ☺).

Il me semble extrêmement important de reconnaitre la qualité des conditions de travail qui nous sont offertes à l'Inria. Je reconnais à quel point elles sont privilégiées et bien meilleures que dans beaucoup d'autres établissements de l'Enseignement Supérieur et de la Recherche (français comme à l'étranger d'ailleurs), même si récemment il a fallu se battre pour les protéger. À ce sujet, je voudrais remercier mes ami·es et collègues de la Commission d'Évaluation et du SNCS-FSU qui font vraiment un travail hallucinant.

Aujourd'hui, la qualité de nos conditions de travail tient grâce au travail sensationnel effectué par les différent·es agent·es dans les services supports à l'Inria et donc je tiens à les remercier également.

Merci à mes parents, qui sont venus de Bretagne pour découvrir ce qu'était qu'une HDR ☺(ça ne s'arrête jamais ces diplômes ?!), et en profiter pour m'aider au niveau logistique et déménagement.

Enfin et surtout, je voudrais remercier Laura et nos enfants Riley et Alison, qui tous les jours me donnent de nombreuses excellentes raisons pour réussir à garder un équilibre sain entre ma vie professionnelle et ma vie personnelle, équilibre qui est souvent dur à trouver dans notre milieu.

[1] Service Expérimentation et Développement

# Contents

# Introduction

In the last 10 years and since my PhD, I have tackled a wide variety of research topics in the theoretical design of system software for High-Performance Computing, particularly in the allocation of various resources depending on the system constraints:

**I/O scheduling** [C2, J9, J3, RR3, C4, C8, C12]: A key issue in supercomputers is the management of data that has been generated and that needs to be stored on disks (checkpoints for resilience, visualization etc). Part of my research has been on scheduling to minimize contention and increase I/O performance of HPC applications.

**Scheduling Stochastic Applications** [J5, C6, C9, C11, C10, J10]: With the advent of HPC, new types of applications are run on supercomputers. Those applications have different characteristics than the typical HPC applications, specifically their execution time is not as deterministic. In this project with researchers from Vanderbilt University, we study Neuroscience application and model their performance using probability distribution. We are interested in the convergence of HPC schedulers and these new applications. In this work we discuss the performance of current schedulers and propose novel algorithmic strategies to deal with stochastic applications.

**Storage-aware Adjoint Computation** [J6, J4, J13, J16]: This is a project with Argonne National Lab (Paul Hovland, Krishna Narayanan) where we study a specific type of graph, backpropagation graphs, and provide efficient solutions to execute them in the presence of storage constraints. With Julien Herrmann we developed a library where those algorithms are implemented (H-Revolve).

**Resilience** [C18, J2, C5, RR2]: Since my PhD I have been interested in managing failures on HPC resources. Checkpointing is the standard technique to protect applications running on HPC (High-Performance Computing) platforms: after each failure, the application executing on the faulty resource is interrupted and must be restarted. Without checkpointing, all the work executed for the application is lost. With checkpointing, the execution can resume from the last checkpoint, after some downtime (enroll a spare to replace the faulty processor) and a recovery (read the checkpoint). Lately I have been working on the limitations of checkpointing models.

The common grounds of these studies is the methodological approach that I bring when tackling them. My reseach has been focusing on proposing algorithmic solutions and evaluating new approaches to these studies. Reading back 10 years of research, I was able to observe the evolution and maturity that I gained on how to model performance of resource management software. This is what I will try to describe in this document.

## Introduction on High-Performance Computing

High-Performance Computers (also called supercomputers) are massive infrastructure allowing the execution of extremely large parallel applications. These applications come from a wide range of domains,

such as material physics, climate modeling/prediction, astronomy etc. Used as a cornerstone of some industrious applications (self-driving cars, drug discovery etc), supercomputing is also one of the pillars of scientific discovery (such as recently the discovery of Higgs Boson, the formation of supermassive blackholes). With the advent of Big Data and machine learning, and the race to Exascale (a supercomputer able to compute at a peak of $10^{18}$Flops) an **explosion of application domains turned to such resources**. In addition, the structure of these machines is changing; the scientific workflows are becoming more complex; their execution patterns are drastically evolving. These resources are extremely expensive, both in terms of construction (the *Frontier* supercomputer, expected to be one of the first Exascale machines, is estimated to **half a billion euros**[2]) and exploitation (Fugaku, the current fastest supercomputer, consumes 28MW, which represents **24 million Euros** per year; which should be added to several important maintenance costs). **Given such a high-cost and energy consumption, utilization of these systems has to be as close to 100% as possible!**

The allocation (or *scheduling*) problem consists in allocating the different jobs (applications) on the shared computational resources based on their requirements. The middleware solution in charge of this allocation is the Resource and Job Management Systems (RJMS), with the most used being the Slurm Workload Manager [1] (running on most of the 500 most powerful supercomputers). Initially, these middlewares focused mostly on the allocation on compute resources (CPU, GPU), but they now start to include the allocation of data resource such as bandwidth and storage. They are central to the performance of HPC centers and as such have become extremely complicated pieces of machinery. Research on the design of job management has always made two key assumptions:

- *User-provided resource needs*: Users are in charge of declaring the volume of resource needed by their applications during their execution;
- *Jobs, not Users*: The RJMS system sees submissions as independent jobs, even though, in practice, an increasing number of jobs are part of a larger user workflow.

**User Provided Resource Needs** RJMS expect users to give them precise information about the behavior of their application at submission time, such as the wall time that is requested by the users. Then RJMS take this information into account to perform resource allocation. This is a simple model, but unfortunately it is naive because it makes the unrealistic assumption that used-provided resource needs are accurate. It has been long documented that user estimates are inaccurate (overestimated) [73, 58, 56], despite user's best effort! This inaccuracy not only hurts the performance of the system [76], but also discourages new users to come to these important resources ("How much time am I supposed to ask for this application, what happens if I overestimate, do I wait longer? Am I penalized?"). The above assumption may have made sense when applications were regular parallel applications, designed by HPC experts, running on homogeneous machines, but the growing diversity of applications and the heterogeneity of supercomputers call for new approaches. In summary, **RJMS ask information that most users are unable to provide accurately, and by using this inaccurate information the overall performance of the supercomputer is negatively impacted.**

**Jobs, not Users** The scheduling decisions of the RJMS are based on existing information at the time of decision: which jobs are available, since when, what resources they need, what resources are available. It also uses information about the share of the platform globally allocated to each user to provide some fairness in resource usage. However jobs are more complicated than this. Upcoming applications embrace more dynamic, heterogeneous multi-phase workflows, where the results obtained by some jobs trigger the submission of new jobs. RJMS and scheduling algorithms barely consider this

[2] https://www.pcmag.com/news/us-to-spend-600-million-on-frontier-exascale-supercomputer

important behavior, critical to user satisfaction and response time. Some tweaks have been introduced (such as the notion of karma or niceness, of job dependencies for workflows), with negative consequence to the point that they had to be deactivated on some large systems [39]. **Currently, because of their outdated designs, RJMS rely on collections of ad-hoc "hacks" to solve problems that were not initially anticipated, resulting in unexpected negative consequences.**

Research on RJMS is extremely active in the field of HPC as the smallest loss in system utilization can cost millions of euros. Yet, this research remains very conservative for the same reasons, focusing on how to perform local improvement, how to include new knowledge, new architectures. Because of these constraints, existing algorithms and optimization strategies have been less than suboptimal. It has also repelled users from those systems. This has been **a major barrier** to the efficient usage and democratization of supercomputer, wasting an enormous volume of resources and energy.

**Content of this document:** I hypothesize that job resource requirements and temporal variations are in essence stochastic. The variability of their needs is inherent and can be large. Based on this hypothesis, I believe that HPC scheduling algorithms and softwares should embrace the uncertainty of job resources requirements. In the design of algorithmic solutions for HPC system softwares, this uncertainty is not taken into account in a satisfying manner.

In this document I make the case for new ways to incorporate the inaccuracy of knowledge into models designed for Resource and Job management systems and HPC system software algorithmic. This demonstration will be based on the research I did in the last ten years.

Through questions and specific examples, I try to show in this document how one can (i) design a model that accurately represent applications, while being *practical*; then (ii) show the importance of questioning the limits of the model proposed, by testing these models against real world hypothesis.

*I purposefully chose to focus this thesis on a discussion on models. However, models are only the first step of the approach; designing new algorithms and scheduling strategies is also at the heart of my contributions. Please refer to the corresponding publications for more details.*

This document is partitioned as follows:

- In Chapter 1, I present some introductory remarks on the design of a model for Resource Management Algorithmic. In particular I discuss the usual hypothesis for model design.
- In Chapter 2, through a specific use-case (fault-tolerance), I show how one can find and evaluate inaccuracy in a model.
- This leads in Chapter 3 to reopening the leading questions on how to design properly a model. In addition, I present my case for better understanding of evaluation objectives in this Chapter.
- In Chapter 4, through the use-case of I/O scheduling, I discussed very thoroughly a search for efficient models. Specifically I show how the models may be related to the optimization objective that one wants to study.
- In Chapter 5, I discuss these questions on another use-case: batch scheduling and the inaccuracy of job runtimes.
- Finally I provide some concluding remarks and open on what I believe will be important research directions in the future.

# Chapter 1

# On Designing an Accurate Model for a Resource Management Problem

Assume a new research question related to system performance in HPC arises. This questions can come from a system administrator complaining about low performance –*e.g. observation of I/O congestion between applications [J9, C23]*–; from a new architecture design being proposed –*e.g. Cache-partitioning is a technology that allows to share exclusively the cache between different processes running on the same node [J7]*–; or simply a new idea using existing techniques –*e.g. "what would happen if, when there is a failure, instead of waiting for a node to be rejuvenated, I killed another, less important job [RR2]?"*–.

## 1.1 Designing our first model

To answer such questions, one needs to formally define the problem. This is what is often called a *scheduling problem* [28]. Most of the time, it needs three distinct elements to be well-defined:

- Elements describing the application;
- Elements describing the machine;
- Elements describing the optimization objective(s).

Describing these elements is the modeling phase of a problem resolution. This phase is then used to get a better understanding of the problem, formalize it explicitly and design algorithmic solutions used for optimization purpose.

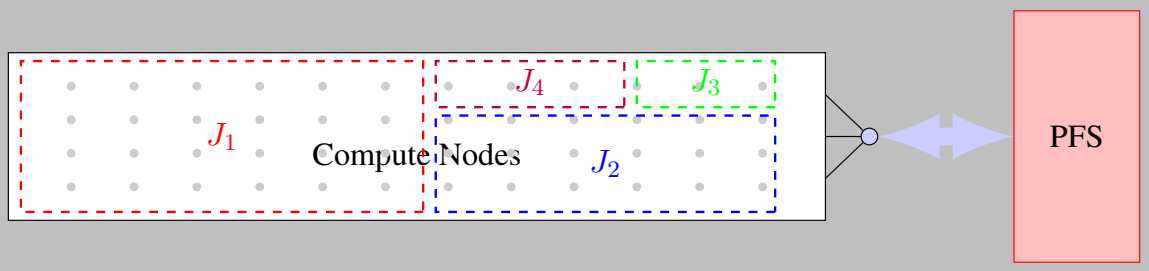
**Example** (Modeling I/O behavior [J9, C4])**. Consider the following issue:** ***I am seeing I/O congestion occurring at the I/O bandwidth level, it slows-down the performance of applications.***

Here we propose a modelization for this problem. It starts by understanding the architecture considered, before proposing a modelization for applications. Finally, if one wants to measure if a solution addresses the problem, one needs to be able to define *performance*.

### 1.1.1 Machine Model

We consider a parallel platform structured as follows: $P$ compute nodes are sharing an I/O node (sometimes called a forwarding node) which is available to perform I/O operations from the com-

pute nodes to the parallel file system. It can send and receive data to/from the Parallel File System with a maximum bandwidth $b$.




### 1.1.2 Applications

We consider a batch of scientific applications that need to run simultaneously onto the parallel platform. Applications consists of a series of consecutive *non-overlapping* phases: (i) a compute phase (executed on the compute nodes); (ii) an I/O phase (a transfer of a certain volume of I/O using the available I/O bandwidth) which can be either reads or writes.

Formally, we have a set of $n$ jobs $\{J_1, \ldots, J_n\}$. Each job $J_i$ requests $Q_i$ compute nodes for its execution. $J_i$ consists of $n_i$ successive, blocking and non-overlapping operations: (i) $W_{i,j}$ (a compute operation that lasts for a time $w_{i,j}$); $V_{i,j}$ (an I/O operation that consists in transferring a volume $v_{i,j}$ of data). Therefore, if the bandwidth available to $J_i$ to transfer its I/O to the PFS is equal to $b$, the time $T_i$ needed for the total execution of $J_i$ is:

$$T_i(b) = \sum_{j \leq n_i} w_{i,j} + \frac{v_{i,j}}{b}. \tag{1.1}$$

However, in general the I/O bandwidth is shared amongst several applications and it may incur delays to the execution of $J_i$.

### 1.1.3 Optimization problem

To measure performance, we can define several objectives. Given a schedule, each job $J_i$ is released at time $r_i$ and finishes its execution at time $C_i$.

The *stretch* $\rho_i$ of $J_i$ is the ratio between the actual execution time and the minimal execution time:

$$\rho_i = \frac{\sum_{j \leq n_i} w_{i,j} + \frac{v_{i,j}}{b}}{C_i - r_i} \tag{1.2}$$

(where $b$ is the maximum available I/O bandwidth). A stretch of 1 means that the application is not impacted by the other applications running on the system. A stretch of 2 means that due to I/O contention, the application takes twice as long to execute as it would normally. Typically the stretch is an objective more *user* oriented. The Dilation $D$ of the system is the maximum of these stretches. It is an objective that one is looking to minimize.

$$D = \max_i \rho_i$$

The *System Efficiency* (SE) of a schedule is the peak performance of the platform, i.e. the number of operations per time units:

$$\text{SE} = \frac{1}{n}\sum_{i=1}^{n} Q_i \cdot \frac{\sum_{j \leq n_i} w_{i,j}}{C_i - r_i} \tag{1.3}$$

It is an objective that one tries to maximize. Typically, the system efficiency is an objective more *platform* oriented.

## 1.2 Evaluating a model

After designing a model of the problem, we need to be able to evaluate its practicality with regards to the objectives that we are targeting. With this in mind, often the first question that one wants to answer is:

**Question 1.1.** How accurately does your model fit reality?

**Example.** Describing an architectural model connected by physical links is something that is *easy* to instantiate. A large pan of research is performed on this topic, and tools such as HWLOC [12] provide ways to represent it and analyze it (Fig 1.1).

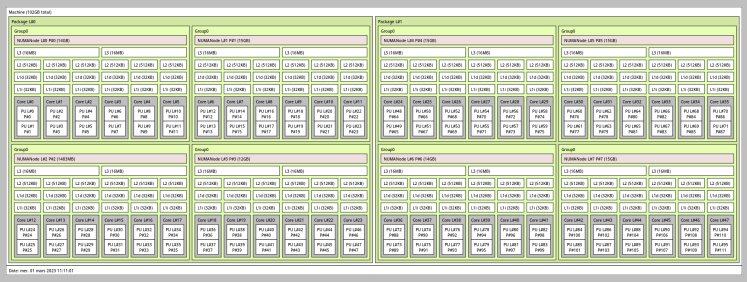


Figure 1.1: Graphical representation of a two processor AMD EPYC 7352 (2x24 core) topology as computed by HWLOC [12], B. Goglin, 2019 (reproduced with authorization).

Yet even then, some behaviors, particularly with respect to bandwidth performance are hard to model correctly. Velho et al. [79] have focused on invalidating certain flow-level models of network communications. They show that one can never be fully confident in such models and assert that when proposing a model, one should study its limitations.

Often the model is not able to accurately represent reality. However, is this an issue? Is Question 1.1 really the question that we want to solve? A model is not an end in itself, but more a mean to design **algorithms** that perform well with respect to our **target objectives**.

The risk on focusing on Question 1.1 is to provide an hyper-parameterized model. In the past, hyper-parameterization has led to the design of many meta-heuristic based strategy, which, when used indiscriminately, often had poor performance compared to heuristic designed for a model with fewer parameters [67].

When considering a model that has too many parameters to describe it, a recent alternative approach has been to consider the model as a black-box function [53, 55, 52]. Similarly, in this case designing

efficient combinatorial algorithms to optimize the objective function is complicated, and researchers often have to rely on reinforcement learning algorithms or generally deep learning algorithms to resolve this problem [9, 72].

But the fact that a performance is dependent on many input parameters does not mean that all parameters are equally important.

**Example.** Isakov et al. [38] studied the impact of several parameters to predict the I/O performance of an application. In Figure 1.2, one can see that information on five key parameters (I/O volume, runtime, cumulative read, write and metadata time) is enough to obtain most of the prediction performance obtainable.

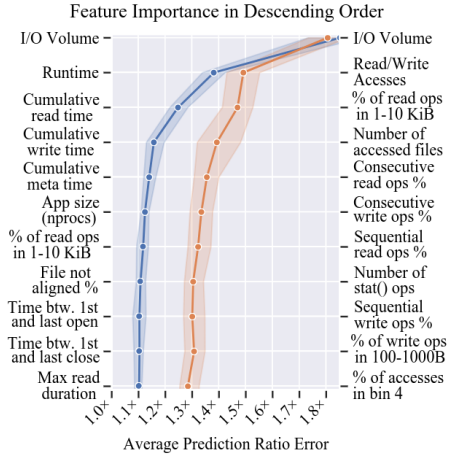



Fig. 5: Permutation feature importance (PFI) [13] of XGBoost models on two datasets. The blue line and labels represent PFI on a dataset with time-based features, and the orange line and labels represent PFI on a dataset without time-based features.

Figure 1.2: Various I/O parameters have various importance in predicting correctly the I/O performance. Figure from [38], reproduced with authorization from the authors.

An alternative example comes from a work with Francieli Boito and Luan Teylo [C2]. In this work we demonstrated the importance of storage target allocation in applications' I/O performance I/O performance (Fig 1.3). In this case, we show that more than the number of storage targets, it is how the targets are balanced amongst the servers that impact the performance.

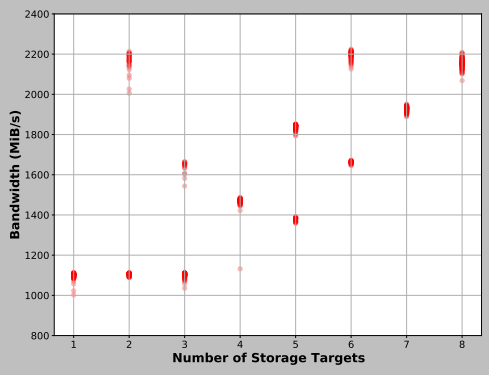



Figure 1.3: The allocation on storage servers plays a major role, and could account for a large variability in bandwidth performance. All details for this evaluation can be found in [C2].

This really underlines the importance of the question *Why are we constructing a model?*, and the fact that a performance model should often be designed in the context of an optimization objective.

Following this remark, in my past research, I have focused on an alternative question:

**Question 1.2.** Given this model, are we able to design algorithmic strategy that perform well on what we are evaluating?

The intuition behind this is that we do not necessarily need a perfect model: some parameters may have negligible impact on an algorithm design compared to other parameters, and hence should not be used in ths design. The measure of performance could be inaccurate, while the behavior of the solution could still be as efficient in the real-world experiment than it is in a simulated scenario (relatively to other solutions).

Of course, simulations based on this model would not be able to predict the exact performance of a real-work evaluation. But what we really need from this model, is the ability to design a solution that performs well (and an easier model may improve the tractability of a problem).

**Example.** Even-though the models are hard to describe accurately, we were able to show that a simple model described in the previous paragraph is enough to predict performance extremely accurately.

Using the model presented in Section 1.1, we designed two algorithms: Periodic, which pre-computes a static schedule based on the job model and Online, which takes online decisions. We measured their performance with respect to two objectives: the system efficiency (which measures how well the system is used), and the dilation, which measures the cost to an individual user. We performed two evaluations: one on a home-made simulator, and one running IOR benchmarks on a development platform. All details can be found in [J9].

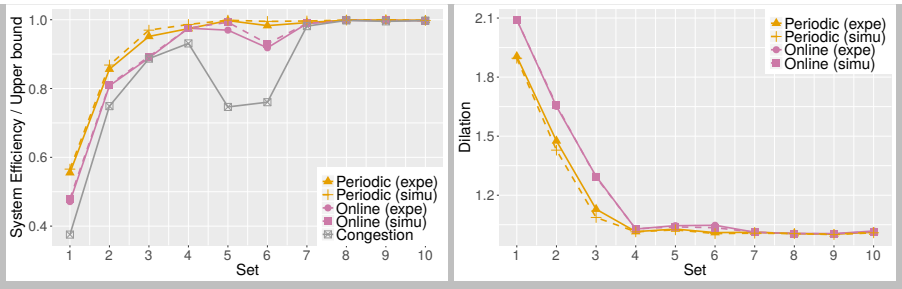



Figure 1.4: Performance for both experimental evaluation and theoretical (simulated) results. `Congestion` is the system measure without additional scheduling strategies. The performance estimated by our model is accurate within 3.8% for periodic schedules and 2.3% for online schedules. The $x$-axis corresponds to different scenarios evaluated, sorted by increasing Upper-bound on System Efficiency (details can be found in [J9, Table 2]).

These algorithms were evaluated on various set of applications, with a performance measured both by simulation (using the model), and experimentally (using I/O benchmarks on a platform that was made available to us). We observe (Fig. 1.4) that the performance of the simulated algorithms are particularly accurate.

When focusing on the accuracy of the performance of an algorithm instead of the accuracy of the model, we were able to verify the coherence of it on several machines: Vesta (a development platform for the super-computer Mira) [C23], Jupiter (a platform at Mellanox) [J9], and Plafrim (a platform at Inria Bordeaux) [RR3].

## 1.3 General comments

When designing a model to represent a scheduling problem, focusing too much on the accuracy of the model may have detrimental consequences, at the point that one cannot design scheduling algorithms anymore. Deep reinforcement learning has been proposed as a solution for this hyper-parameterization of models: we replace the algorithm-designer by a machine learning algorithm.

It is not clear that such precision in a model is needed. The question that one should ask from a model is not whether it is accurate, but whether this model allows for the selection of the most efficient strategy.

If it permits this, then one can say that the model is *robust* [3, 31]: given the inaccuracy of the model, the expected practical performance are still within a certain precision. This *robustness* is a trade-off between its accuracy and its practicability This trade-off can sometimes be measured and we study this on a use-case in Chapter 2.

### On the importance of designing a model and the cost of experiments versus simulations

Designing a model to understand performance behavior has many advantages:

1. It helps architecture designers to understand the limits of an architecture from the application perspective. A RL algorithm may not show that the limitation of a system for a given workload is due to memory performance (for instance).

2. It is key to the design of a simulation based evaluation. For instance in a recent I/O work [RR3], we were able to show that 80% of the experimental results that we performed were within 3.5% of

the performance predicted by a simulator, describing cases where the simulations were accurate and cases where they were not. This is even-though we have seen in Figure 1.2 that I/O needed an extremely large number of parameters to be accurate. In practice the experiments took us 44h of compute-time on a HPC machine for a simulation time of 5s. The complete evaluation that we performed to study the limits of our solution would not have been doable by experiments only.

# Chapter 2

# Finding and Evaluating Inaccuracy in a Model: Use Case Fault-Tolerance

In this Chapter, to illustrate the difficulty in finding and evaluating inaccuracy in models, we discuss the use-case of fault-tolerance. In a nut-shell, fault-tolerance arises in supercomputing from the plentiness of computing resources: even if each computing resource has a very low probability of failure, the failure probability of the whole HPC system is much higher. *With Yves Robert, we wrote a very nice introduction to failures and fault-tolerance techniques which I highly recommend ☺ [O3].*

To deal with failures, several techniques exist [34, O3]. In this Chapter I will focus on Checkpoint-Restart: jobs are periodically checkpointed. When a failure occurs, the work done since the last checkpoint is lost, and the job restarts from the last checkpointed chunk of work (see Figure 2.1).

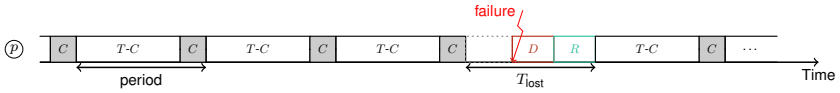



Figure 2.1: An execution.

## 2.1 Modeling fault-tolerance mechanisms

Following Section 1.1, we present a model for the unreliable platform and the applications (and fault-tolerance mechanism). Then we present an objective to optimize.

For the modelization step, we use one of the simplest model from the literature for periodic checkpoint [84, 20]: a job can be checkpointed at any time (divisible job), the failures are independent and their Inter-Arrival Times (IATs) follow an exponential distribution.

### 2.1.1 Unreliable platform model

We consider a parallel platform subject to failures. We assume that the failure inter-arrival times are IID (independent and identically distributed) and follow an Exponential distribution $\text{EXP}(\lambda)$ of parameter $\lambda$, whose PDF (Probability Density Function) is $f(x) = \lambda e^{-\lambda x}$ for $x \geq 0$. The MTBF is $\mu = \frac{1}{\lambda}$. When hit by a failure, the platform is unavailable during a downtime $D$.

### 2.1.2 Job model

We consider a job of length $\text{TIME}_{\text{base}}$ when there are no failures. We consider the job to be *divisible*, i.e. it can be checkpointed anywhere at a constant cost $C$. In case of a failure, it takes $R$ units of time (after the downtime $D$) to recover from the last checkpoint.

### 2.1.3 Optimization function

The typical objective that one measures is the *Waste* of resource usage:

$$\text{WASTE} = \frac{\text{TIME}_{\text{final}} - \text{TIME}_{\text{base}}}{\text{TIME}_{\text{base}}} \tag{2.1}$$

where $\text{TIME}_{\text{final}}$ is the total time of the execution.[1]

### 2.1.4 Optimal solution

For periodic solution, the optimal solution for this problem is what is called the Young/Daly formula [84, 20]: when $C$ is small before $\mu$, then one should checkpoint with a period of $\sqrt{2\mu C}$ to minimize the waste.

There has been a large body on literature on how to optimize this formula for different contexts [34]. We showed that Daly actually made a small mistake in his model [J19], and that the right approximation for his model should be $\sqrt{2(\mu - (D + R))C}$, but the impact is generally negligible, and the simple satisfying formula is a good approximation.

In the next sections, we discuss several examples where this model does not take into account real system constraints along with its robustness.

## 2.2 Inaccuracy in architecture: the case for independence of failures

*The interested reader can find more information about the science discussed in this chapter here [C18].*

The well-known Young/Daly formula for the optimal checkpointing period [84, 20] is valid only if failure inter-arrival times (IATs), are IID (Independent and Identically Distributed) random variables.

### 2.2.1 Does the model correspond to reality (Question 1.1)?

There are several ways that this model can be inaccurate. We will discuss here the IID hypothesis.

Several studies [51, 74, 6] have intuited correlation in failures. Indeed, the intuition is that when something causes a failure (for instance a component overheats), it can create several other failures, in *cascade*. This assumption may impact the MTBF of the platform (more failures take place), but also strongly impacts the independence hypothesis between failures.

Intuitively, if a failure increases the chance of having another failure soon, a fault-tolerant strategy may want to enter a *degraded* mode and checkpoint more frequently after a failure.

[1] Note that the ergodic theory says that the time average is almost surely the space average, hence for a long enough execution, this is equal to the expectation of the waste.

**Example** (Evaluating the presence of *cascade* failures [C18])**.** We aim at providing a quantitative answer to the following question: to what extent are failures temporally independent? We base our analysis on publicly available failure logs from LANL [45, 42] and Tsubame [78].

In a nutshell, the method that we proposed analyzes the distribution of pairs of two consecutive IATs. Intuitively, consider a failure log and its IATs. If small values are scattered across the log, we do not conclude anything. On the contrary, if a small value follows another small value, we may have found the beginning of a cascade. Our approach checks the frequency of having two consecutive small values, and compares this frequency with the expected value when IATs are independent.

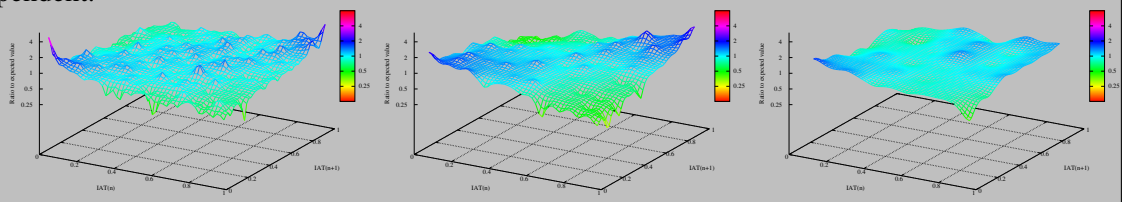


Figure 2.2: Lag plot for LANL2 (left), LANL 20 (center), Tsubame (right) [C18].

| Log | Number of Faults | Presence of Cascades |
|---|---|---|
| LANL 2 | 5351 | Yes |
| LANL 16 | 2262 | Inconclusive |
| LANL 18 | 3900 | Inconclusive |
| LANL 19 | 3222 | Inconclusive |
| LANL 20 | 2389 | Maybe |
| Tsubame | 884 | Inconclusive |

Altogether, the observation is that there are indeed some cascades, albeit not very frequent, in some failure logs. Hence we were wrong to assume failure independence everywhere.

### 2.2.2 Are the solution designed for the simpler model able to perform well (Question 1.2)?

Does the previous result mean that when modeling a system, we need to model *cascade* failures? This would start to get extremely complicated as we would need to model various type of origin for failures and their possible impact.

**Example** (Specific algorithmic strategies to deal with the presence of *cascade* failures [C18])**.** In a second step, we can evaluate the usefulness of cascade-aware checkpointing algorithms. For this, we have used both public and synthetic logs. We used the latter to explicitly create "artificial" cascades. We have shown that current cascade-aware bi-periodic checkpointing algorithms are not really more efficient than the standard periodic checkpointing approach that considers failures to be independent. Finally, by using a brute-force search over all possible bi-periodic algorithms and considering omniscient oracles that know exactly when cascade failures will strike, we have shown that only insignificant gain should be expected from designing future cascade-aware checkpointing algorithms. The conclusion is that we can wrongly, but safely, assume failure independence!

Note that there has been other results that show the same results, for instance the failure inter-arrival time may not follow an exponential distribution. Tiwari et al. [74] and Heien et al [33] confirmed the observation that the Young/Daly formula is a very good approximation for Weibull distributions.

This conclusion stresses the fact that having an extremely accurate model is not necessarily critical. Of course the work done is important (even if it is not rewarding): one has to evaluate it thoroughly to be able to come to this conclusion.

## 2.3 Inaccuracy in job model

In order to derive their optimal formula, Young and Daly considered a model where the job consisted in a single continuous execution which could be checkpointed anywhere (divisible) [84, 20].

### 2.3.1 Does the model correspond to reality (Question 1.1)?

Being able to checkpoint anywhere is possible using some system-based library (such as FTI [7]), but the checkpoint cost may be extremely expensive (a complete memory footprint), and restart may be a lot harder for the job. In practice, many scientific applications are decomposed into computational *iterations*, and where one can (should) checkpoint only at the end of an iteration. Indeed, for iterative applications, checkpointing is efficient, let alone possible, only at the end of an iteration, because the volume of data to checkpoint is dramatically reduced at that point. A wide range of applications fits in this framework. Iterative solvers for sparse linear algebra systems are a representative example [61, 60].

Some authors have considered including these considerations in order to *improve the model* (answer to Question 1.1) to include a limited number of possible checkpointing locations.

The model becomes [O3]:

*Given a linear chain of $n$ tasks, $T_1, T_2, \ldots, T_M$. Each task $T_i$ has weight $w_i$. The cost to checkpoint after $T_i$ is $C$, and $R$ to recover from this checkpoint.*

The problem of finding the optimal checkpoint strategy for a linear chain of tasks (determining which tasks to checkpoint), in order to minimize the expected execution time, has been solved by Toueg and Babaoglu [75].

In practice, large scale applications often have extremely large number of iterations ($M$) and this number may even be unknown (iterate until convergence). Hence this algorithm may be inapplicable. In addition, there may be multiple possibilities to checkpoint within an iteration, each with different cost.

**Example** (Model for an iterative application with multiple tasks per iterations [J2])**.** We consider an iterative application $\mathcal{A}$. Each iteration of the application consists of $n$ parallel tasks $a_i$, where $0 \leq i < n$, task $a_i$ has length $t_i$ and memory footprint $M_i$. We define the length of an iteration as $T = \sum_{i=0}^{n-1} t_i$. We assume that the number of iterations is extremely large (and unknown).

Note that the hypothesis of a large number of iterations model has an impact on the criteria evaluated: there is no guarantee that the limit of the slowdown of a schedule exists when the number of iterations tends to infinity, hence we focus on the limit of the upper-bound of this slowdown. *The interested reader can find more information in [J2].*

Overall, given this model, we are able to provide two important results:

**Theorem 2.1.** *There exists a periodic schedule that is optimal.*

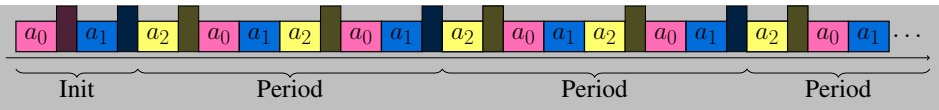



Figure 2.3: A periodic schedule represented graphically for a job where each iteration consists of three tasks $a_0$, $a_1$, $a_2$ [J2].

**Theorem 2.2.** *We can compute an optimal periodic schedule in polynomial time.*

The complexity of the algorithm to compute the optimal schedule is: $O\left(n^7 \log(\sum_{i=1}^{n} t_i)^2\right)$. It relies on the proof of an upper-bound on the size of the period, followed by a brute-force search in periods smaller than this bound via a dynamic programming algorithm to evaluate the minimal slowdown. For visual representation, the algorithm that computes the optimal period is represented in Figure 2.4 (without going into the details of the notation since this is not the goal of this work).

**Algorithm 1** Finding the minimum slowdown of a pattern of size at most $2n^2(k^\star+1)$

**procedure** PATTERN($k^\star$, $n$)
  $maxK \leftarrow 2n(k^\star+1)$
  **for** $i_0 = 0$ to $n-1$ **do** ▷ Initialization of ProgDyn
    **for** $\ell = 0$ to $2n^2(k^\star+1)$ **do**
      **for** $b = 1$ to $n$ **do**
        **if** $\ell = 0$ **then**
          $\mathcal{C}_{\min}(i_0, \ell+1, \ell, b) \leftarrow 0$
        **else**
          $\mathcal{C}_{\min}(i_0, \min(maxK, \ell+1), \ell, b) \leftarrow \infty$
      **for** $k = 1$ to $\min(maxK, \ell)$ **do**
        $\mathcal{C}_{\min}(i_0, k, \ell, 0) \leftarrow \infty$
  **for** $i_0 = 0$ to $n-1$ **do** ▷ Precompute $\sum_{i=1}^{k} t_{i_0+i[n]}$
    $W[i_0, 1] \leftarrow t_{i_0+1[n]}$
    **for** $k = 2$ to $2n^2(k^\star+1)$ **do**
      $W[i_0, k] \leftarrow W[i_0, k-1] + t_{i_0+k[n]}$
  **for** $\ell = 1$ to $2n^2(k^\star+1)$ **do** ▷ Computing the ProgDyn
    **for** $i_0 = 0$ to $n-1$ **do**
      **for** $k = \min(maxK-1, \ell)$ downto 1 **do**
        **for** $b = 1$ to $n$ **do**
          $\mathcal{C}_{\min}(i_0, k, \ell, b) \leftarrow \min(\mathcal{C}_{\min}(i_0, k+1, \ell, b),$
$\mathbb{E}_\lambda(W[i_0, k], C_{i_0+k[n]}, R_{i_0}) + \mathcal{C}_{\min}(i_0 + k[n], 1, \ell-k, b-1))$
  $SD = \infty$ ▷ Computing the minimal slowdown.
  $T = \sum_{i=1}^{n} t_i$
  **for** $i_0 = 0$ to $n-1$ **do**
    **for** $m = 1$ to $2n(k^\star+1)$ **do**
      $SD_{\text{temp}} = \mathcal{C}_{\min}(i_o, 1, mn, n)/mT$
      **if** $SD_{\text{temp}} < SD$ **then**
        $SD \leftarrow SD_{\text{temp}}$
  **return** $SD$

**Theorem 6.** PATTERN($k^\star, n$) *(Algorithm 1) returns the slowdown of the pattern of an optimal periodic schedule with time complexity* $O((k^\star)^2 n^5)$.

Figure 2.4: The optimal algorithm for the precise job model. Notations are not introduced on purpose but the full details can be found in [J2].

In the end, one can see that focusing on Question 1.1 and having a more accurate model is at the trade-off of a more complicated solution.

*Note that in practice there are many ways in which this model can still be unsatisfying, but you get the idea ☺.*

### 2.3.2 Are the solution designed for the simpler model able to perform well (Question 1.2)?

The solution obtained by refining the job model provided in Figure 2.4 has (at least) two main drawbacks. The first one that one may think of is its important computational complexity. The second one (which is to me the most important drawback) is its lack of simplicity compared to the elegant Young/Daly formula.

Indeed, as one can see from Figure 2.4, there are many indexes that are likely to create coding errors, and a quite complicated proof that make it hard to verify[2].

This is also why, I believe that one must really focus on Question 1.2: is it worth having this extremely complicated algorithm?

**Example.** In order to evaluate the importance of this solution, we compare to various simple reference strategies:

1. One that checkpoints after each task;
2. One that checkpoints after each iteration;
3. One that works at least $\sqrt{2\mu\bar{c}}$, where $\bar{c}$ is the average checkpoint cost, after the previous checkpoint and that checkpoints as soon as possible;
4. One that works at least $\sqrt{2\mu c_{\min}}$, where $c_{\min}$ is the smallest checkpoint cost, after the previous checkpoint and that checkpoints the first task with a checkpoint cost of $c_{\min}$.

We evaluated these strategies compared to the optimal solution in various applicative and failure scenarios, and the conclusion is that overall, the third strategy (average Young/Daly) generally performs within 2% of the optimal strategy!

In the end, I believe that our work [J2] is important. It allows to assess the almost quasi-optimality of easier greedy strategies! However no-one should ever implement the algorithm that we proposed ☺.

## 2.4 General comments

In this chapter, we have tried to demonstrate that focusing on the accuracy of a model can be harmful to the design of algorithmic solutions. One should keep in mind that designing a model serves a purpose: finding an efficient algorithmic solution.

Focusing on the performance of the model-based algorithmic solution can help to provide simpler solutions which do not necessarily perform worse in practical scenarios.

[2] Truthfully, I am not 100% sure that the reviewers have verified the proof thoroughly or if they have mostly trusted us for the details.

# Chapter 3

# Towards More Practical Models?

As discussed in the previous chapters, it is not uncommon to make simplifications when designing a model. It allows to derive more easily theoretical results such as performance guarantees or simply algorithms that are easier to implement in practice, while still providing good performance in the "real" environment. These simplifications can be considered voluntary.

## 3.1 How accurate can the input to instantiate your model be?

Throughout my recent research I have focused on a limitation of these models that is more deceitful: What happens when the job model is indeed correct, but the information that one can hope to obtain to instantiate this model is inaccurate?

Probably the most famous example from the literature is the case of runtime estimates. It does not seem extravagant to model a parallel HPC job as follows: a number of processors $p$ and an execution time $t$. It however becomes a problem when the scheduling strategy uses this information to take its scheduling decisions. Many heuristics however do. For instance, it is known that a good heuristic for scheduling single node jobs to minimize the total execution time is to sort them by decreasing execution time and to schedule them greedily (list-scheduling). On the contrary, to minimize the average response time, one wants to sort them by increasing execution time and schedule them greedily (list-scheduling).

We discuss this example in more depth in [C1].

**Example.** For fault-tolerance strategy, we have shown that using the Young/Daly formula, i.e., checkpointing with a period of $\sqrt{2\mu C}$ where $\mu$ is the Mean Time Between Failures (MTBF) and $C$ the checkpoint cost, is a good approximation for the optimal strategy.

But how does one know the MTBF $\mu$? How do we instantiate the model? Obtaining the MTBF of a platform can be done in several ways such as:

- Based on maker data, when all machines are taken independently.
- Based on historical data once your machine has run for a long enough time.

In practice, neither of these strategies guarantees an accurate platform MTBF. Gupta et al. [30] showed that the MTBF of a platform was subject to large variations.

We can actually measure the cost of inaccuracy. Given a platform whose real MTBF is $\mu_0 = 40$min (ground truth, data considered to be unknown). We can evaluate the performance of the algorithm that takes a MTBF $\mu$ as input and checkpoints every $\sqrt{2\mu C}$, and compare its performance to the algorithm that checkpoints every $\sqrt{2\mu_0 C}$ (Fig. 3.1).

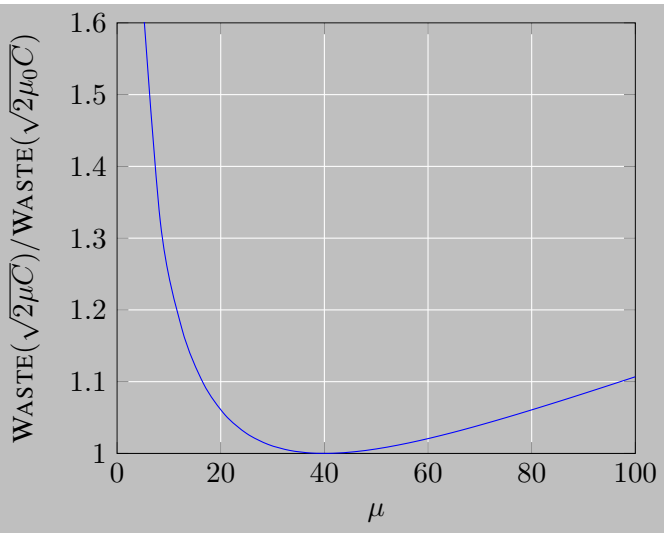



Figure 3.1: Ratio of the waste (Eq.(2.1)) of the algorithm that uses an inaccurate MTBF $\mu$ compared to that of the algorithm that uses the exact MTBF $\mu_0$ ($C = 3$min, $\mu_0 = 40$min). As one can see, the cost of misevaluating the MTBF of the platform by a factor of 2 ($\mu = 2\mu_0$ or $\mu = \mu_0/2$) incurs a 6% increase in waste. Hence interestingly we can see that the Young/Daly formula is quite robust to input inaccuracy.

To sum up, one of the limitation of Question 1.2 is that we focus on what the job is, instead of focusing on what information can be provided to us (by the user, an analysis tool, historical traces etc).

There are many cases where system software expects a specific type of input, either from a user, or from a predictive system, but where there is very little chance for the input to be correct.

Hence, building on Question 1.2 I believe that a more interesting question that one should answer when designing a model for resource management software is the following:

**Question 3.1.** Given this model **and given the expected quality of the information obtainable**, are we able to design algorithmic strategy that perform well on what we are evaluating?

Of course it is extremely hard to define properly what is *the expected quality of the information obtainable*: one may expect it to be dependent of the available technical advances at time $t$.

In the following Chapters, we discuss two examples on how one can try to answer this question.

## 3.2 How to evaluate performance correctly?

A final question that I would like to raise in this Chapter on model design is about evaluating the performance of a solution. In Question 3.1, I discreetly proposed the sentence: **that perform well on what we are evaluating**. But what are we evaluating? And how do we decide? The optimization criteria is an important part of the design of a scheduling problem.

For theoretical results (complexity or approximation results), it is very common to focus on a single criteria [28]. However when studying the performance of a resource management software, optimizing with respect to a single criteria may have unforeseen consequences.

Several optimization criteria are used to evaluate the performance of a Resource and Job Management Software. In this Section, we discuss more in depth those objectives, particularly in the context of High-Performance Computing. We explain their limitations in this context. Part of a work with Robin Boezennec and Fanny Dufossé [C1].

The analysis presented in this work is targeted for High-Performance Computing: building a machine able to perform ExaFlops targets the execution of large scale applications mostly and the validation of the performance of a solution should reflect this. Extreme-scale platforms have a high operating cost and are expected to be utilized as much as possible.

Analysis of HPC system traces showed that *Users are now submitting medium-sized jobs because the wait times for larger sizes tend to be longer* [58]. To execute medium-size jobs, it is probably more efficient (cost-wise) to have multiple smaller clusters than an HPC machine with a dense interconnect.

In order to define the objective we propose the following notations for job $J_i$:

| | |
|---|---|
| $r_i$ | The release time of job $J_i$ |
| $C_i$ | The completion time of job $J_i$ |
| $t_i^{\text{real}}$ | The real length of job $J_i$ |
| $t_i^{\text{wait}} = C_i - r_i - t_i^{\text{real}}$ | The waiting time of job $J_i$ |

### 3.2.1 Mean (bounded) slowdown

The mean slowdown (also called mean flow) is the main optimization criteria in many recent works on improving resource management in HPC [48, 14, 86]. Its goal is to provide a measure of fairness over applications.

The slowdown $S_i$ of job $J_i$ (also called the flow of the job) corresponds to the ratio of the time it spent in the system over its real execution time. Formally, it is defined as

$$S_i = \frac{t_i^{\text{real}} + t_i^{\text{wait}}}{t_i^{\text{real}}} = \frac{C_i - r_i}{t_i^{\text{real}}}$$

Note that in practice many jobs are extremely small (few seconds). In these cases their slowdown could be arbitrarily high even if their wait time is ridiculously small (a five minutes wait time for a job that dies instantly (one second) would result in a slowdown of 300).

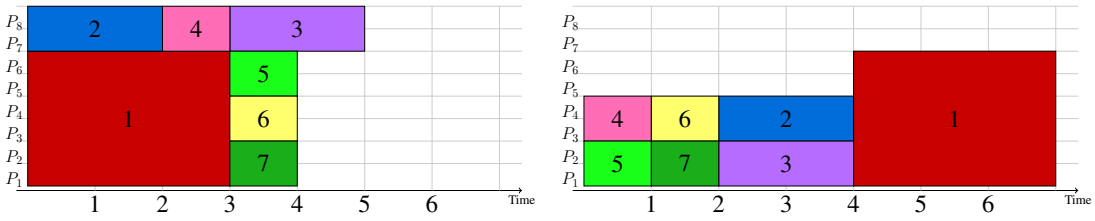
The solution that is often used is to consider a variant of the slowdown called the *bounded slowdown*:

$$S_i^b = \max\left(\frac{C_i - r_i}{\max\left(t_i^{\text{real}}, \tau\right)}, 1\right) \tag{3.1}$$

where $\tau$ is a constant that prevents the slowdown of smaller jobs from surging. Then the mean bounded slowdown $\bar{S}$ is:

$$\bar{S^b} = \frac{1}{n}\sum_i S_i^b, \quad \text{where } n \text{ is the number of jobs}$$

**Limits for HPC workloads** By improving the quality of service to the small jobs, one can considerably improve this objective. This is often what is actually measured when work study this objective, and is the opposite of what a system administrator of an HPC machine is looking for. This is illustrated in Figure 3.2.




(a) Mean slowdown $\approx 2.8$ (b) Mean slowdown $\approx 1.8$

Figure 3.2: In this example, all jobs are released at $t = 0$. Despite what appears to be a more efficient strategy, the left schedule has worse mean slowdown than the right schedule.

We also show that this is subject to a high variability, and hence the performance is highly dependent on the input data.

**Alternative approach** To understand the actual behavior of the system, some work [RR2] consider the bounded slowdown as a function of the size of the job. In this case, this objective is not one to optimize anymore, but more a qualitative way to measure and understand the performance of a solution. Another approach is to use a weighted version of the average slowdown where large jobs are given more weight than smaller jobs.

### 3.2.2 Utilization

This optimization criteria measures how fully the platform is occupied. It is a particularly important objective for an HPC platform that costs multiple-million of dollars yearly to operate. This is the main objective studied in [24, C10, C11].

If $W(t_1, t_2)$ is the total amount of work done between $t_1$ and $t_2$ on a platform with $N$ nodes, the utilization $U(t_1, t_2)$ on the interval $[t_1, t_2]$ is measured as:

$$U(t_1, t_2) = \frac{W(t_1, t_2)}{N \cdot (t_2 - t_1)}. \tag{3.2}$$

Note that when jobs fail to complete fully (for instance because their walltime is underestimated), it is interesting to measure the "useful utilization", i.e. the volume of computation that lead to a successful execution [RR2] .

**Limits for HPC workloads** One of the main limitation is for machines with lower submission rate (i.e. that are not "packed"), then any scheduling solution has the same (low) utilization since it corresponds to executing almost all jobs during the whole window. Utilization by itself does not allow to discriminate between different schedule qualities (Figure 3.3).

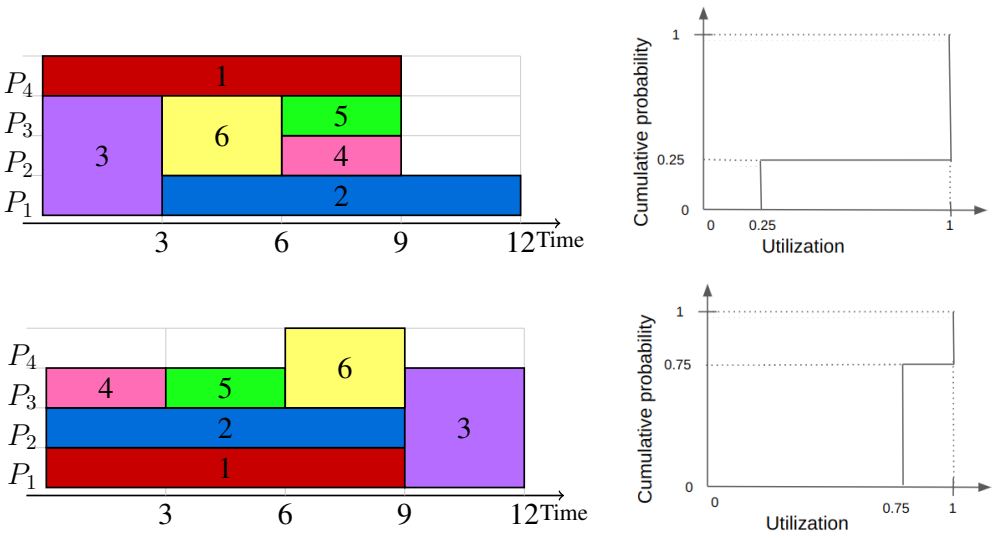



Figure 3.3: Two examples of schedules and their corresponding cumulative density function. Even though the global utilization is the same between the two schedules (13/16), the distribution of their utilization differ significantly.

Another one is the fact that it is more a system administrator target: how to maximize the yield of my machine. It does not give a sense of the quality of the schedule: an easy way to maximize utilization would be to have a large queue of jobs waiting to be executed and find the one that works best at all time (often favoring smaller jobs that can fill a hole).

**Alternative approach** Our observations show that in some scenarios if the utilization of an HPC platform is lower than 93%, the "quality" of a scheduler has no impact on the average utilization of the schedule.

There are settings where the workload has different "modes" (such as intensive in the day; low on requests in the night), in this case it may be interesting to study utilization of these workloads separately. Having a good understanding of one's workload is important.

We found that a way to measure this is to study the density function for the utilization (see Figure 3.3). Indeed intuitively, for two identical job submission schemes a "better" scheduling algorithm will have more phases at very high utilization (and hence more at lower utilization). Indeed, it can pack jobs as soon as they are available, whereas a poorer scheduling quality will delay jobs from phases of time with intensive job counts to phases with less intensive job counts.

When two schedules have an almost identical utilization (because all jobs are scheduled in the same time window), we propose to measure the variance of the utilization as a way to differentiate the quality of a schedule: the "best" algorithm from a utilization perspective should have a higher variance: more time-windows with very high occupation and more time-windows with low occupation. For example the schedule at the top of Figure 3.3 is better at using all available resources at the same time, leading to a variance of utilization 9 times greater than the schedule at the bottom of Figure 3.3.

**Some remarks on utilizing the variance:**

1. This metric allows to qualify whether one schedule is better than another one from a utilization perspective, but it lacks interpretability: what does having a variance $x$ times greater than another one means overall? This is an open question for us.

2. It is important to note that the variance is only relevant to compare schedules with a similar utilization. If it is not the case, one can just tell which schedule is better by looking at the utilization.

### 3.2.3 Response time (and Wait Time)

Mean response time (or mean wait time) is a metric often used in the literature [24, C11, 77, 56, 71, 85]. The response time $\mathcal{RT}_i$ of a job $J_i$ is the duration between the submission of the job and its completion, or equivalently its wait time and its length.

$$\mathcal{RT}_i = t_i^{\text{wait}} + t_i^{\text{real}}$$

The mean response time is equivalent to the mean wait time since the difference is the mean runtime which depends on the workloads but not on the schedule. In the following, we only address the response time, but our reasoning identically apply to wait time.

**Limits for HPC workloads** One of the first concern about response time is that it does not differentiate between the waiting time and the size of the job, hence a job of 1 hour that waits for 1 minute has the same response time as a job of 1 minute that waits for 1 hour.

In addition, using this objective gives equal importance to all jobs, independently of the work they represent. In an HPC workload, this gives an advantage to the numerous "small" jobs, even if they only represent a very small portion of the workload. In Figure 3.2 we can see that the schedule on Fig. 3.2(a) intuitively looks more efficient than the schedule on Fig. 3.2(b). Yet it has a worse mean response time (3.6 vs 3). This is because schedule 3.2(b) favors small jobs despite being less effective at densely packing jobs.

Similarly to the mean bounded slowdown, we show in [C1] that when using a workload from a big compute center (with an important variability in jobs sizes), the mean response time is mainly going to be influenced by the proportion of very small jobs which are backfilled (and then have a very low response time). In addition to not corresponding to what we want to optimize, the relative performance between different algorithms is also subject to a lot of variability depending on the workload. This is something that was confirmed by our experiments and which is covered more in depth in [C1].

In the end, this is a limit for the response time objective because simply improving it does not necessarily mean improving the quality of the overall schedule (from an HPC perspective at least).

**Alternative approach** Goponenko et al. [27] have used the weighted mean response time, where one weights the response time by a priority (such as the total amount of work of a job, or the number of nodes that a processor uses). We argue that when computing the average response time in the context of an HPC job scheduler, one should give a higher weight to bigger jobs. For example giving a weight proportional to the area of the job would allow to transform the mean response time in a system administrator metric: as Goponenko et al. [27] underlined, using this weight would mean swapping a job with two smaller jobs of the same duration but half the resources would not change the weighted response time. This way neither small nor big jobs are favored, and what is measured is the ability of the scheduler to densely pack jobs. Alternatively, Gainaru et al. [C11] have proposed to only measure the response time of non-backfilled jobs.

## 3.3 General comments

To conclude this Chapter, there are two take-away that I believe should be considered when designing a model for HPC system software algorithmic:

1. We should not only consider what the *objects* (jobs, machines) are, but consider whether we can instantiate them accurately. Note that if we cannot instantiate these models accurately, it does

not mean that the model is necessarily wrong; it does however mean that one should study the robustness to instantiation inaccuracy of the solutions designed.

2. When studying an optimization objective, one should not only focus on a numerical value, but understand the limits of such objectives.

Many objectives when optimized have negative side effect for large scale platform (such as improving the performance of small jobs at the detriment of large jobs).

Yet a significant body of work, particularly recent work that discuss improving batch-scheduling techniques using machine learning still optimize these objectives. As an example, recent research directions have focused on using RL-based scheduling in batch schedulers [85, 86]. They show that by using RL into the batch scheduling, one can improve considerably the response time and bounded slowdown at a small cost in utilization.

By simply looking at the objective function, their analysis lacks quality elements that could show the limits of their performance as discussed in the previous section. Specifically it is very likely that their important improvement in response time or slowdown are mostly used by the improvement of the slowdown/response time of small jobs, which may be done at the detriment of those of larger jobs.

Work by Carastan-Santos et al. [14] where the ML algorithm provides a priority function confirms this intuition and the fact that learning-based batch schedulers with the objective of bounded slowdown simply give higher priority to small jobs. Similarly, Legrand et al. [48] have realized the importance of small jobs for bounded slowdown and focus on having an oracle which guesses which job is small and which is large. This is sufficient for important performance gains for this objective.

# Chapter 4

# Finding the Right Model: Use-Case I/O Management

In this Chapter, we take the example of I/O management to illustrate the wide possibility of model selection for the same *use-case*. For the rest of this Chapter, we consider the architecture discussed in Section 1.1. Throughout this Chapter we consider applications that perform I/O synchronously, i.e. that do not overlap compute operations and I/O operations. This chapter is based on a five year project funded in part by the ANR JCJC Dash (young researcher funding by the French National Research Agency), and a European H2020 project Admire. It was a collaboration with many people (by alphabetical order): Olivier Beaumont, Francieli Boito, Lionel Eyraud-Dubois, Ana Gainaru, Emmanuel Jeannot, Valentin Lefevre, Luan Teylo, Nicolas Vidal.

**Context on I/O management** As High-Performance applications increasingly rely on data, the stress put on the I/O system has been one of the key bottleneck of supercomputers. The I/O bottleneck is usually defined comparing the speed of the growth of computational power of supercomputer, compared to the speed of the growth of Parallel File System bandwidth.

This becomes an issue when multiple concurrent applications request access to the I/O bandwidth simultaneously: I/O congestion creates delay in the execution of applications, hurting the utilization of the platform.

Many solutions have been proposed to face the congestion issue. Some solutions trade-off storage for computation: for instance compression will reduce the volume of I/O sent to PFS at the cost of extra operations (such as compression, decompression, correction). If data has a limited lifespan, local management is a solution to reduce the data sent to PFS, at the cost of extra space occupied on fast storage that could be used to increase the speed of the computation.

Other solutions generally try to manage the I/O accesses. Amongst approaches that manages I/O accesses, the first one, which is the topic of this chapter is I/O scheduling [C23, 88]: it consists in deciding algorithmically which applications get priority when too many applications are requesting I/O simultaneously. Architectural solutions such as Burst-Buffers [68] allow to smooth the I/O requests over time, reducing the chance of an I/O peak, and allowing to perform I/O almost asynchronously. However, there still remains an I/O scheduling problem [C8] of selecting when the burst-buffers should be emptied to PFS (write operations), or when the data should be prefetched to PFS and moved to the buffers (read operation).

## 4.1 How accurately does your model fit reality (Question 1.1)

In most of my past work [C23, J9, C8, J3], the job model I have used to design I/O scheduling heuristics is the one presented in Section 1.1, that we can name *task model*. A job is represented by a sequence of compute and I/O phases (read and writes) of various length and volume.

**Example** (Example of Application model considering reads and writes [C8])**.** In this work with Olivier Beaumont and Lionel Eyraud-Dubois, we have considered the following constraints. Applications consist in a sequence of up to three consecutive actions: (i) data fetching from disks (read); (ii) computations (compute); and (iii) data uploading on disks (write).

Formally, application $\mathcal{A}_k$ is released at time $r_k$ and consists of $n_k$ iterations. Iteration $i \leq n_k$ of $\mathcal{A}_k$ consists of three consecutive *non-overlapping* phases: a read phase, where $R_{k,i}$ denotes the volume of data read, at read bandwidth $b_k^r$; a compute phase, where $l_{k,i}$ denotes the compute time; and a write phase, where $W_{k,i}$ denotes the volume of data to be written at write bandwidth $b_k^w$. We assume that the phases cannot be overlapped for a given application: reading must be finished before the computation can start, and similarly the computation must be finished before starting to write. This constraint is representative of many applications, whose memory requirements prevent to fetch data for the next phase in advance when the data for the previous phase still occupies the memory.

In practice, $b_k^r$ and $b_k^w$ depend on the resources allocated by the batch scheduler and are given for $\mathcal{A}_k$. Hence, an application can be written as:

$$\mathcal{A}_k = (r_k, b_k^r, b_k^w, \Pi_{i=1}^{n_k} (R_{k,i}, l_{k,i}, W_{k,i})) . \tag{4.1}$$

**Does this model fits reality?** In order to evaluate the model, one needs to instantiate jobs and architecture. A strategy that we use to evaluate our models, is to create a machine simulator that follows the model designed in the work, and that is instantiated with some key parameters from a real machine. For the application instantiation, it is common to use I/O benchmarks (such as IOR [37]). We then compare the performance of the algorithms in the model-based simulator to the performance measured on the actual machine.

**Example.** An example of evaluation is the one that we have performed for [J9]. In there we have proposed a wide variability of applicative scenarios with various performance.

We evaluate various scheduling strategies (online and offline) for each of these job scenarios. We perform the evaluation in three steps: first we simulate behavior of applications and input them into our model to estimate both Dilation (Section 1.1.3) and SysEff (Eq. (1.3)) of our algorithm and evaluate these cases on an actual machine to confirm the validity of our model. *The interested reader will find all details of the evaluation in [J9].*

The results are shown in Figure 1.4, the main observation being that the performance estimated by our model is accurate within 3.8% of that of the real execution. This confirms that this model, with this level of approximation can be considered to be a good fit for I/O-scheduling analysis.

### 4.1.1 Discussion on model-requirements and limits

In order to derive the previous experiments, we used a controlled experimental platform. The applications were not real applications but I/O benchmarks which we instantiated manually. To do so, we

defined the length of the *compute* phases (which are actually *not-doing-anything* phases in the case of an I/O benchmark), and the volume of I/O that is transferred during the *I/O phases*.

Under these constraints, we have seen that if we are able to describe accurately the application (and its I/O patterns), then the task based model allowed to design solutions whose simulated behavior accurately represent what would happen in a real scenario.

However the requirements for task-based schedules are extremely precise: it is not given that for a *real* application it is easy to predict ahead of time the exact I/O volume needed by a phase. Indeed, characterizing the I/O of an HPC application is a challenging task and often requires detailed modeling approaches. The presence of I/O variability due to various reasons including PFS congestion and slow I/O, can make this tasks even difficult.

In addition, the HPC I/O stack only sees a stream of issued requests, and does not provide I/O behavior characterization. Notably, the notion of an *I/O phase* is often purely logical, as it may consist of a set of independent I/O requests generated during a certain time window, and popular APIs do not require that applications explicitly group them.

Hence, some approaches have been proposed to provide high-level aggregated metrics — the most popular example probably being Darshan [65], but on the other hand, these aggregated metrics do not properly represent the *temporal* behavior of applications [83].

**Example.** As a simple example we have shown that simply answering the question: *"Is the I/O of my application periodic or not?"* is hard [RR1].

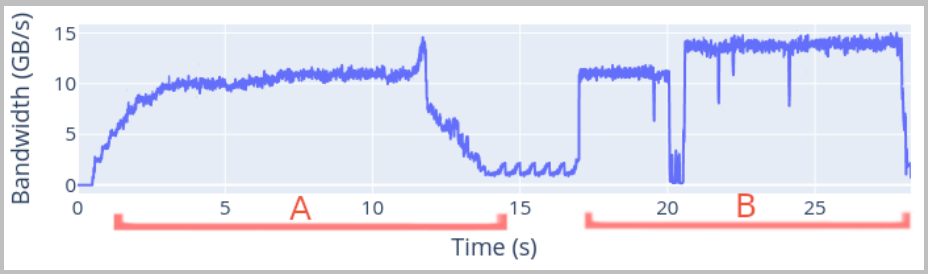



Figure 4.1: Difficulty of detecting I/O phases: Where does A finish? Is B one or two I/O phases? How about A and B together? [RR1]

The first complication is where to draw the border of an *I/O phase* (see Figure 4.1), as it is composed of one or many *I/O requests*, issued by one or more processes and threads. For example, an application with 10 processes may access 10 GB by generating a sequence of two 512 MB requests per process, then do compute and communication phases for a certain amount of time and perform a new 10 GB access. In this case, we need a way of saying that the first 20 requests correspond to the first I/O phase, and the last 20 to a second one. One could propose an approach where the time between consecutive requests, compared to a given threshold, determines whether they belong to the same phase or not. But then a suitable threshold must be chosen that will depend on the system. Moreover, the reading or writing method can make this an even harder challenge as accesses can occur, e.g., during the computational phases in the absence of barriers. Hence, the threshold would not only be system-dependent but also application-dependent.

## 4.2 Measuring the impact of the trade-off Simplicity-Precision (Question 1.2)

The task-based model is accurate, but as we have seen is extremely hard to instantiate (if it is even possible) as it often considers the full I/O pattern to be known. It is also much harder to use it to derive theoretical results [J3]. Furthermore, deriving, a priori, an optimized solution based on theoretical I/O values may not be robust.

In another work [C12] we have proposed a much simpler probabilistic model that can be instantiated very easily. We present this much simpler model in this Section. It is purposefully inaccurate to derive algorithmic solutions. We discuss its limits when evaluating it.

**Example** (Probabilistic-based model [C12])**.** In order to obtain theoretical results, we model application data transfers with a random process. To achieve this, we omit the *phase* behavior in the model. Instead, we consider discretized time units and we assume that during each of these time units, application $\mathcal{A}_i$ sends data with probability $p_i$ (with bandwidth $b_i$). In order to have a time unit corresponding to the characteristic size of the system, we set it as the average value of the application I/O transfer times.

Therefore, in our model, all applications share a common time unit, and there is no correlation between what happens at time step $t$ and $t+1$ (memoryless property). This assumption is of course crucial to build a Markov chain model. However, if the length of a data transfer for $\mathcal{A}_i$ is much longer than the time unit, the fact that $\mathcal{A}_i$ is involved in a communication at time $t$ strongly influences the probability that it is involved in the same communication at time $t+1$. On the other hand, if the period of the pattern for $\mathcal{A}_j$ is much shorter than the time unit, then the I/O bandwidth consumed during one time unit with our model is very imprecise, since it is either sending or not sending during the whole time unit, whereas such an $\mathcal{A}_j$ actually performs several communication and computation phases. **These two examples show clearly that this probabilistic model does not correspond to reality.**

This simple model of application allowed us to build a Markov-chain-based model of the system. Using this, we were able to quickly answer issues about dimensioning of the system, such as for a given set of applications, and for a given Burst-Buffer size and bandwidth, how often does the buffer overflow. Using this, we expect that system administrators can evaluate various complex burst-buffer management strategies. As an example, we showed that waiting until the buffer was at least 20% full before flushing it to disks had almost no impact performance-wise (we call this *Lazy emptying*). This can be useful if one does not want to fragment too much their data, or if there is important latency when accessing the PFS.

**Limits** This simple model is advantageous when it comes to deriving algorithmic policies. If described correctly, these policies can be used in practice, even if the job model is incorrect: for instance
"`flush a buffer only when it is at least 20% full`"
is a policy that can be implemented independently of the behavior of the jobs.
"`Allocate a buffer of size` $S$ `for application` $A$"
(where $S$ is computed through our solution) is another policy that can be used even if the model used to compute it is incorrect.

Should they be used in practice is a different question, and for this we need to evaluate whether the model still makes some sense, even if it does not correspond to reality.

**Example.** One way to evaluate the importance of the simplicity made in the model is to point-out the main hypotheses that were made and to compare the theoretical performance with those on a model closer to real life. Here, we propose to evaluate some of the main limitations of the model: the hypothesis that applications share similar characteristic time (the common time unit proposed in the model), and its memoryless property.

Methodologically, we used data from APEX workflows [46] to generate various application profiles, and compared the I/O performance as predicted by our Markov chain, and the one measured on the discrete event simulator validated in our previous work (that we completed with a burst-buffer model) [C23].

The evaluation confirmed the inaccuracy of the model (see for instance Figure 4.2 and [C12, Section VI.C] for more results and a more thorough analysis).

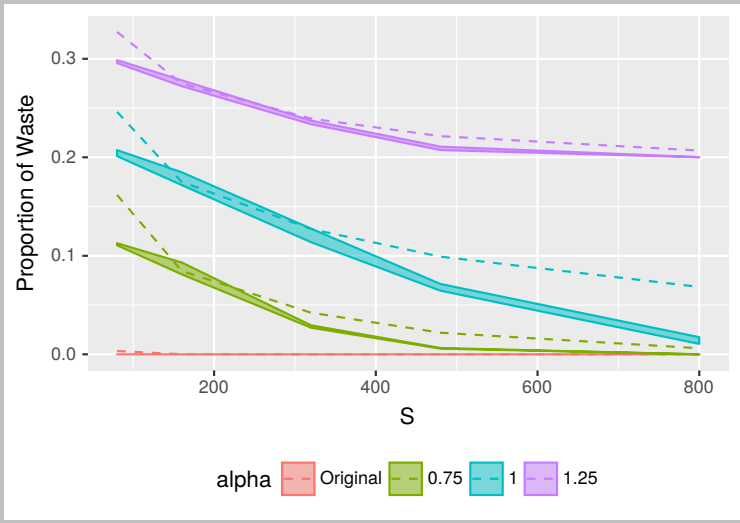



Figure 4.2: Comparison of predicted idle times between Markov chain (dashed lines) and discrete event simulator – APEX data (full lines). $S$ is the size of the buffer (relative to the characteristic time of the evaluation), $\alpha$ is a measure of the stress on the I/O system (higher=more stress). For a complete and thorough definition of the notations from this figure see [J9].

However it does not mean that everything should be thrown away. It shows that the results obtained with this simple model are still somehow representative of the real performance and can be used as a first-order solution. These results show heuristically that the memoryless property tends to be pessimistic with respect to the performance (of course one would need more thorough analysis to confirm this intuition).

This pessimism and the fact that lazy emptying with a 20% threshold seem to have no impact on the Markov model performance [C12, Figures 6 and 7] is also a good indicator and what encouraged us to make this first-order recommendation.

## 4.3 Towards an instantiable I/O model (Question 3.1)

Approach from Section 4.1 [C23, J9, 88] can be called *Clairvoyant* approaches as they consider the I/O patterns of each application are known before-hand. In this case, one can compute a schedule

that optimizes the right objective functions (often maximum system utilization or a fairness objective between applications). This task can be computationally expensive because applications can have a large number of I/O phases. Moreover, these techniques require information that in practice is often not accurate or even accessible (Section 4.1.1).

On the contrary, in Section 4.2, the approach was *non-clairvoyant* since it was based on a probabilistic model. *Non-clairvoyant* schedulers do not know when applications will perform I/O. These accesses are thus simply scheduled given a priority order, the most common being first-come-first-served, or semi round-robin (the I/O served is that of the application that performed I/O the least recently) [87, C23].

Going further, we would like to design an I/O scheduling solution that uses some information about applications, but as little as possible, while remaining robust to inaccuracies in this information.

**Example.** If your I/O scheduler is able to give you aggregated information such as: the execution lasted for 2h, the cumulative I/O was 200TB, and that there were roughly 60 iterations (compute-I/O) [RR1], can we do better than the performance of the probabilistic model?

Hence, to answer Question 3.1, we propose to work with an average behavior and focus on algorithmic design that is robust to inaccuracy. A theoretical example is provided on Figure 4.3, where the "real" behavior is represented in red, and the model selected in blue, with the average behavior $w_{\text{iter}}$ being the total execution length divided by the number of iterations.

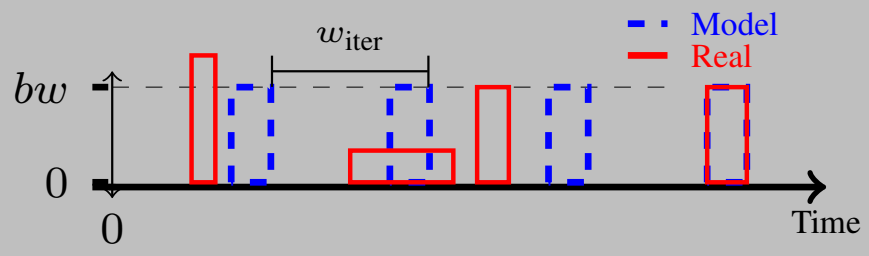



Figure 4.3: An average model for a corresponding “real” behavior.

### 4.3.1 Algorithmic design

For the algorithmic designs, an observation is that when one knows exactly the I/O pattern of applications (*Clairvoyant*), then an exclusive bandwidth access (i.e. one application performs I/O at the time), or semi-exclusive when applications cannot use all the available bandwidth by themselves (i.e. some applications run concurrently using as much bandwidth as they can) seems to be the better choice.

However, in our work [J3, Figure 3] we observed that sometimes without clairvoyance, fair-share (i.e. the I/O bandwidth is shared equally by all applications in a best-effort fashion) behaves better than exclusive heuristics.

With Francieli Boito, Luan Teylo and Nicolas Vidal, we have tried to understand this more in depth for the algorithmic design [RR3].

#### Fairshare vs Exclusive

We illustrate this with a simple example. Consider two applications, (i) one that has relatively large I/O phases (we will call it *large*), which when running in isolation performs periodically:

computation during one unit of time and I/O during one unit of time; and (ii) a *small* one, which in isolation performs periodically: computation during one unit of time and I/O during 0.01 unit of time. We consider two scenarios in Figure 4.4: two *large* applications competing for the I/O bandwidth, and a *large* with a *small*. In the following, we assume accesses, once started, cannot be interrupted, and that, at the beginning of the first I/O phase, the *large* application's request arrived before.

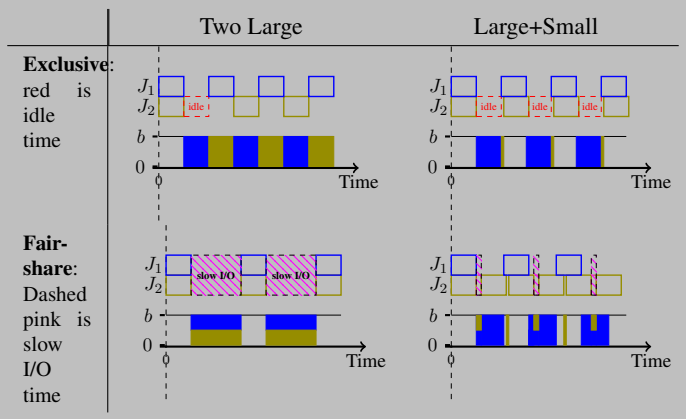



Figure 4.4: Two different pairs of jobs are shown side by side, and two schedulers are shown one above the other. For each of the four parts of the figure, the top half represents activity on compute nodes, and the bottom half the I/O accesses to the PFS (the height represents the portion of the bandwidth used). [RR3]

As one can see from this figure, fair-sharing can be inefficient: for the TWO LARGE scenario it takes three units of time to perform the work that exclusive does in roughly two units of time (after initialization). However, the opposite can also be true: in the LARGE+SMALL scenario, exclusive takes roughly two units of time to perform the computation of the small application when fair-sharing can do it in roughly one, with almost no extra cost for the *large* application.

### 4.3.2 Bandwidth sharing

We argue that sharing the bandwidth does not have to be done fairly. In Figure 4.5, we consider the two application profiles from the previous section, *small* and *large*. If the small I/O phase finishes before the large one, then giving it more of the available bandwidth improves *locally* (i.e. for this phase) the performance of *small* without delaying the *large* one.

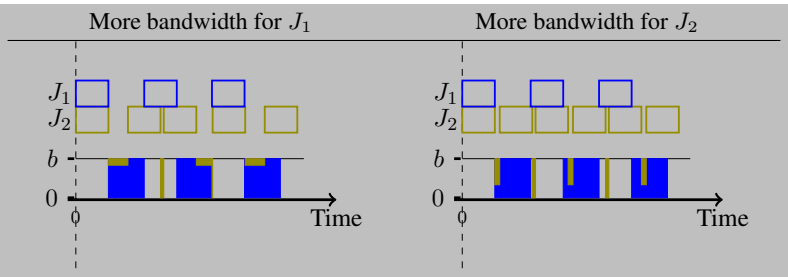



Figure 4.5: The applications share the bandwidth, but $J_2$ (the one with small I/O phases) receives more or less of it than $J_1$ [RR3]

Of course, improving the performance of the small application may *globally* delay the large one, however the impact seems negligible. For instance, in Figure 4.5 (on the right), at every two iterations of $J_1$, $J_2$ performs an additional I/O phase. Hence $J_1$'s I/O phase becomes $1\%$ longer. Over its execution, this is a slow-down of less than $0.25\%$. By comparison, when $J_1$ receives most of the bandwidth, every three iterations, one *small* I/O phase takes $0.9$ units of time instead of $0.01$. In this case, three iterations of $J_2$ take $4.1$ units of time instead of $3.2$, i.e. a slow-down to the *small* application of roughly $28\%$.

### 4.3.3 High-level presentation of an algorithmic strategy

Based on these various motivational examples, we proposed a *Set-based approach* [RR3]:

- Each applications performing I/O is assigned to a set $S_i \in \{S_0, S_1, \dots\}$.
- Each set $S_i$ is allocated a bandwidth priority $p_i$.
- At any time, only one application per set is allowed to do I/O (exclusive access within sets). We use the first-come-first-served (FCFS) scheduling strategy within a set (i.e. we pick the application that requested it the earliest).
- If applications from multiple sets request I/O, they are allowed to proceed and their share of the bandwidth is computed based on the priority given to its set.

Proposing a heuristic for this approach consists therefore of answering two important questions: (i) how do we choose the set in which an application is allocated, and (ii) how do we define the priority of a set.

We have then evaluated a heuristic that schedules jobs in each set depending on the order of magnitude of their mean time between consecutive I/O phases (which we call their "characteristic time" ($w_{\text{iter}}$ in Figure 4.3). The priority given to the set is then rougly the inverse of their order of magnitude. The more technical details can be found in the paper [RR3].

Through an extensive evaluation, we have shown excellent performance of this heuristic on various challenging scenarios. Notably, it was shown to be quite robust to inaccurate information, due to being based on an average estimation instead of precise application information. Finally, we have provided insights on how our method can be implemented.

## 4.4 General comments

To conclude this chapter, I would like to stress that I do not think that there is a *right* model. As we have seen here, for a given problem there can be a multitude of modeling strategies. What we should keep in mind at all time is the motivation for constructing the model, the fact that they should be adapted to the problem that one is studying, particularly in terms of designing solutions. This is what is stressed out by Question 3.1.

# Chapter 5

# Finding the Right Model: Use-Case Resource Management

One of the most famous example for the importance of input accuracy is the case for runtime estimates in resource management software.

## 5.1 A simple model for job representation (Question 1.2)

Algorithmic design in Resource and Job Manager Software (RJMS) often requires the users to submit an allocation value (how many resources does the job need) and a time limit/walltime (how long is the run expected to last). They use this information to take scheduling decisions often in a greedy way (typically Best-Fit algorithm / List-scheduling heuristic). However, it is a widely acknowledged fact that runtime estimates are overestimated [17, 58, 77]. As was again very recently highlighted by Patel et al. [58] in their analysis of Mira's log (supercomputer at Argonne), the wait-time of users has increased significantly in the past 10 years. In their work, Tang et al. [70] showed that improved runtime predictions can decrease the average wait time and slowdown by up to 20%. Note that some authors have tried to show that this inaccuracy may be beneficial to the system, but their work has been debunked in depth [77, 76].

**This is an excellent example for the limits of model realism as one would study if only considering model design Question 1.2.**

Over time, the impact on the system performance has been described by several researchers [76]. Incentives have been proposed to users to try to improve this estimate with very little effect by Lee et al. [47]. Several approaches have been made to predict reliably the execution time of applications. Tsafrir et al. [77] proposed to use a greedy approach that takes the average of two last actual runtimes of the user as a prediction. Machine learning-based solutions have been proposed [82] to try to improve the prediction, but while some gains could be observed, those are still unsatisfying precision-wise. In addition, those solutions often forget to describe the impact on the number of under-estimations [24] of those estimate. Under-estimation of walltime has a critical impact: the jobs are interrupted if they are longer than the walltime estimate, and user then need to pay for the execution of their jobs without getting the result.

**New designs of RJMS** Faced to this challenge, several solutions have been proposed. The historic solution is backfilling which consists in scheduling small applications in the gaps created by the overestimation, under the condition that they do not delay existing reservations [56]. Backfilling is still under heavy evaluation, recently Carastan-Santos et al. [15] have shown by studying several metrics

that, in order to improve backfilling algorithms, one should prioritize jobs according to a smallest area first criterion instead of the usual first-come-first-served.

Another approach to cope with the fact that information is unreliable is the use of Reinforcement Learning (RL) to design RJMS [66, 85, 23]. Reinforcement learning (RL) is a type of machine learning technique where agents learn efficient policies through interaction with their environment. Preliminary results on very limited scenarios seem to show some improvement, but those solutions are still very immature to be able to see whether they can or cannot be a real solution. Limitations of RL include its scalability [29, 23], the input it requires [23], and its implementation [23]. In addition, an important issue with some of these solutions when they are hyperparametrized and unstructured is their *black box* nature. The lack of explainability of those solutions may limit their widespread dissemination.

## 5.2 New application models that include input (in)accuracy (Question 3.1)

In the rest of this Chapter, we discuss how one would construct a model based on the model design Question 3.1. *Part of this Chapter is taken from a recent work with co-authors: Gainaru, Goglin and Honoré (one of my former PhD Student) [J5].*

What if the model (number of processors and estimated length) considered from the start was wrong? Could we change the model considered to take into account the inaccuracy expected from the data, and would this help us to design better algorithms? These are the questions that we discuss in this Section.

**Example.** Field in close relation to HPC have proposed new models that try to incorporate inaccuracy for the execution time of a job.

**New models in Probabilistic Timing Analysis** Real-Time Systems (RTS) have been precursors in realizing the importance of coming up with other representations for program execution times [22]. The historical study in RTS has focused on the Worst-Case Execution Time (WCET) of a program, studying upper-bounds on the execution time that could be used in systems with hard or soft deadlines. Since the years 2000, a part of this community has studied probabilistic Worst-Case Execution Time (pWCET), a grandeur that models a probabilistic upper-bound on the performance of the program. Several methods have been provided to describe the pWCET (detailed in [22]). Roughly, these rely on a mix of static analysis of the program [5, 4], statistical estimates [19, 62] and benchmark/evaluation [63, 2]. Some of these techniques will certainly be useful to study HPC applications, however because of the respective dimensions of the applications under consideration they will not be applicable directly. Indeed, real-time analysis often considers program running on few nodes for short amount of time, far from the volumes of computations needed for HPC. In HPC, a coarse-grain analysis without access to the code may then be sufficient. Similarly, the timing constraints are different: instead of tight-constraints, HPC could target average-case behavior. Hence instead of the design of probabilistic upper-bounds with guaranteed, heuristic representations of the behavior could be sufficient. The models would trade-off the guarantees from RTS for precision over generic behavior.

**State of the art in Theoretical Scheduling** Dealing with the scheduling of jobs whose execution time is unknown has also been an important problem of the scheduling community. One of the most common approach has been the design of *robust* solutions: determining the schedule

with the best worst-case performance compared to the corresponding optimal solution over all instantiations of job processing times [21]. This is sometimes also called a min-max regret approach. The typical job models used in robust scheduling are ones where the uncertainty on the processing times is represented by either a discrete set [40] or a continuous bounded interval [25]. Because these models are very generals, the typical results are not practical (negative results on approximability even in the single machine case for different objective functions [54]). Some authors (such as budgeted uncertainty [8, 10]) showed that given more structured job models, one could obtain positive results (such as approximation algorithm). Yet, all these models are cursed to target large competitive bounds (at best) because they deal with general instances and adversarial proofs may construct very biased running time distributions. In addition, intuitively robust solution guarantee performance in a worse case scenario and so are important for real-time constraints, but do not guarantee expected performance, which are more important for HPC systems. Extensions to robustness have been introduced such as *Recoverable robust* solutions [50], where the goal is to guarantee robustness with bounded recoverable means (such as swapping at most $k$ times pre-determined allocations). This provides again interesting directions but have very limited impact in the research of practical solutions for HPC job scheduling.

In *Stochastic Scheduling* [57], jobs are modeled as random variable. The main difference compared to deterministic scheduling is that one aims to show the optimality of a scheduling policy with respect to the expectation of an objective function. There are few cases in the literature for which optimal scheduling policies are known to be efficiently computable. This is considerably more difficult. Several authors have focused on the performance of variants of deterministic algorithms where one uses the expected processing time instead of the deterministic processing time, to study the expectation of usual objective functions (such as expected makespan, response time, weighted completion time) [81]. To be able to obtain results, authors often include very constrained conditions (such as exponential distributions [13], same general distribution between jobs [80]). More recently, approximation algorithms for stochastic machine scheduling have been derived [64], but those are complicated (using mix of linear programming based stochastic scheduling policies) and inapplicable for HPC because of their high computational complexity and general performance.

These last examples are interesting with respect to the trade-off correctness of the model versus applicability: it seems that in the theoretical scheduling community, models that are realistic with regard to what input data could be expected performed loosely.

Note that in my recent work [C9, C6] I showed that we were able to obtain quasi-optimal stochastic solutions for HPC-oriented objectives for any distributions. The algorithms used are quite approachable with low complexity. This path has opened up many directions. This is what I discuss in the rest of the Section.

### 5.2.1 Stochastic job model: Case study of a Neuroscience Application

On purpose, this Section goes in a lot more technical depth than the other section. It shows how one can experimentally construct a new type of model for HPC job scheduling. It is motivated through a thorough study from an upcoming HPC application from neuroscience: SLANT. First high-level observations are made, then explained with lower-level performance analysis.

#### Spatially Localized Atlas Network Tiles (SLANT)

The study of this work is centered around a specific representative neuroscience application: SLANT [36, 35]. This application performs multiple independent 3D fully convolutional network (FCN) for high-

resolution whole brain segmentation. It takes as input an MRI image obtained by measuring spin–lattice relaxation times of tissues. We use a CPU version of the application[1]. There exists different version of SLANT depending on whether the network tiles are overlapped or not. Here, we consider the overlapped version (SLANT-27 [36]) in which the target space is covered by $3 \times 3 \times 3 = 27$ 3D FCN. The application is divided into three main phases: i) a preprocessing phase that performs transformations on the target image (MRI is a non-scaled imaging technique) ii) a deep-learning phase iii) a post-processing phase doing label fusion to generate the final application result. Each of the tasks may present run-to-run variations in their walltime.

### High-level observations

In recent work [J10], observations showed large variations in execution time of neuroscience applications, complicating their execution on HPC platforms. We are interested in verifying and studying this. To do so, we run SLANT on 312 different inputs. These inputs are extracted from OASIS-3 [43][2] and *Dartmouth Raiders Dataset* (DRD)[3] [32] datasets. We run the application on a *Haswell* platform composed of a server with two Intel Xeon E5-2680v3 processors (12 core @ 2,5 GHz). We run the docker image presented in the Git repository of SLANT-27 using the Singularity container runtime.

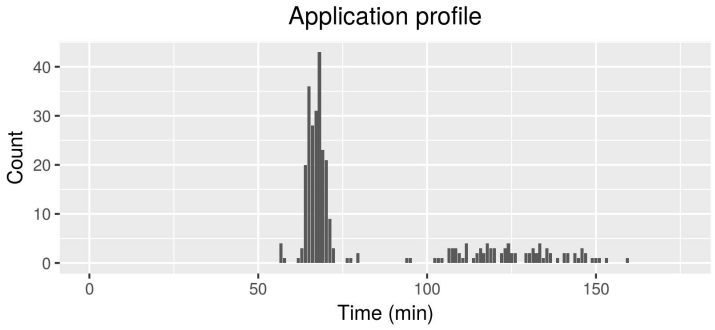



Figure 5.1: SLANT application walltime variation for various inputs.

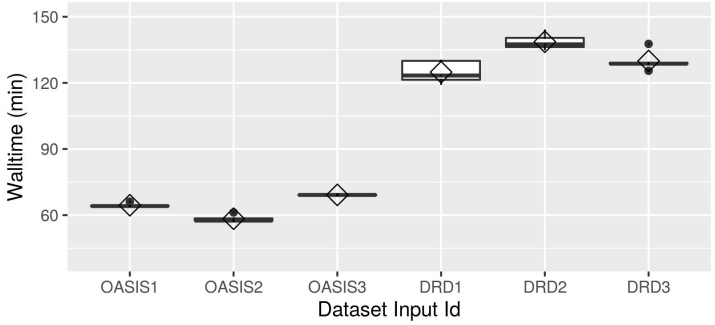



Figure 5.2: Performance variability on identical inputs. Variability is studied over five runs.

In Figure 5.1, we confirm the observations about the large walltime variations. Specifically we can see two categories of walltimes which correspond to the two datasets: OASIS inputs have a walltime of

[1]The code is freely available at https://github.com/MASILab/SLANTbrainSeg

[2]For this very large dataset, we only used a subset of available data.

[3]Available at http://datasets-dev.datalad.org/?dir=/labs/haxby/raiders

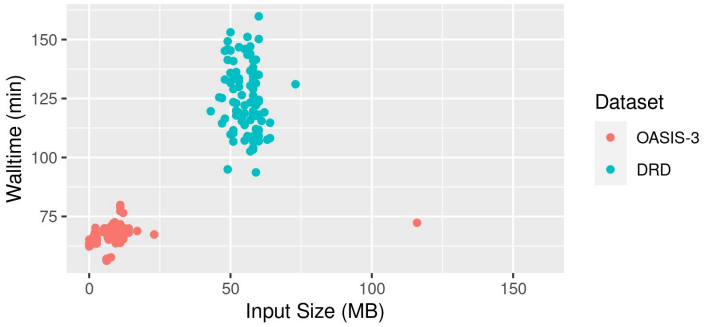



Figure 5.3: Correlation between the size of the input and the walltime over the 312 runs.

70min$\pm$15% and DRD inputs have a walltime of 125min$\pm$30%. The natural questions that arise are the following:

- Is the walltime variation due to a machine artifact (or is it related to the *quality* of the input)?
- Is the walltime variation due to the input size (and can it be predicted using this information)?

We study these questions in the following experiments. First we randomly select three inputs of both datasets and execute them five times each. We present the results in Figure 5.2. We see that the behavior for each input is quite stable. There are slight variations for DRD inputs, but nothing of the order of magnitude observed over all inputs. Hence, it seems that the duration of the execution is mainly linked to the input.

We then study the variation of walltime as a function of the input size in Figure 5.3. We can see that for a given dataset, the walltime does not seem correlated to the input size. The corresponding Pearson correlation factors are 0.30 (OASIS) and $-0.15$ (DRD). The datasets however seem to have different input types: except for the outlier at 120 MB, the input sizes of OASIS vary from 0 to 30MB while those from DRD vary from 45 to 75MB. We present visually the type of inputs for the two databases in Figure 5.4. Intuitively, the performance difference on OASIS versus DRD inputs is probably due to the resolution quality.

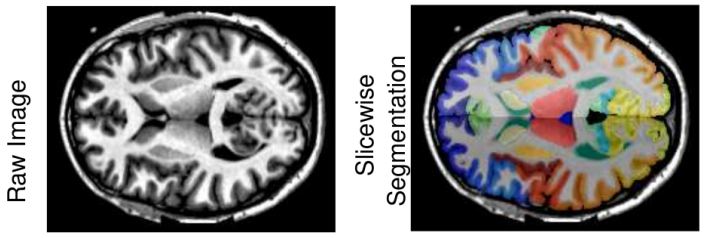
Altogether, we believe we can give these preliminary observations on these new applications:

1. We confirm the observations of significant variations in their walltime.
2. These variations are mostly determined by elements from the input, but are not correlated to the size of the input (*quality* and not quantity).

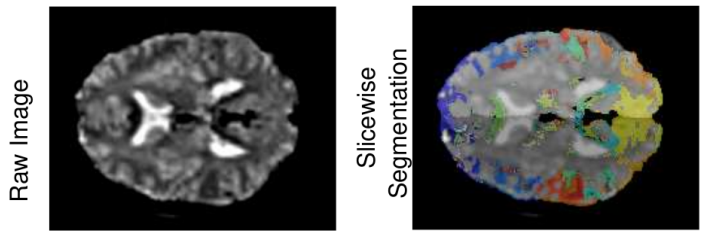
### Task-level observations

Studies using machine learning methods to estimate the future resource consumption of an application assume a constant peak memory footprint (e.g. [69]). In this section, we study more closely the memory behavior of these new HPC applications.

Figure 5.5 presents the memory footprint of two runs of the SLANT application, one for each of the input categories. Note that all other runs follow similar trends, specifically **the peak memory usage is not dependent on the input**, only the time depends (and hence the average memory utilization). For




(a) Segmentation for OASIS.




(b) Segmentation for DRD.

Figure 5.4: Typical inputs and outputs based on the dataset.

both profiles, we can see clearly the three phases of the application (pre-processing, deep-learning, post-processing). Note that these traces hint at the fact that the difference in executed time is more linked to a quality element since there is fewer pre/post-processing time for OASIS input.

In the following, we focus our discussions on the runs obtained from the 88 DRD inputs (Figure 5.5b) because their pre/post processing steps are more interesting, although the same study could be done for the OASIS inputs.

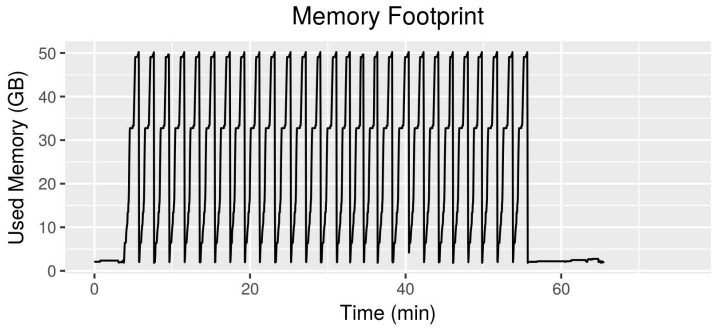
These memory footprints show that the runs can be divided into roughly seven different tasks of "constant" memory usage:

- *pre-processing* phase: This phase includes the four first tasks. The 1st task shows a memory consumption peak of around 3.5GB for the few first minutes of the application execution. The $2^{nd}$, $3^{rd}$ and $4^{th}$ tasks have respectively a peak of about 10GB, 6GB and 10GB.
- *deep-learning* phase: The $5^{th}$ task, represents the deep-learning phase. This task presents a periodic pattern with memory consumption peaks going up to 50GB. Each pattern is repeated 27 times, corresponding to the parameterization of the network tiles in SLANT-27 version.
- *post-processing* phase: The $6^{th}$ and $7^{th}$ tasks model the last phase of the application, with a memory peak to respectively 3.5GB and 10GB.

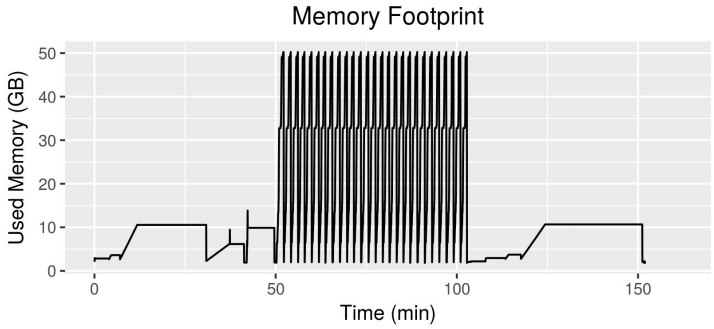
In the second step of this analysis we are interested in the behavior of the job at the task level. We decompose the job into tasks based on the memory characteristics by using a simple parser (see Figure 5.6). This parser returns the duration of each task within each run based on their memory footprint. Note that this decomposition can be incorrect, we discuss this and its implications later.

Using the decomposition in tasks, we can plot the individual variation of each task execution time (for simplicity, we only considered execution time at the minute level) in Figure 5.7.

We make the following observations. First, all tasks show variation in their walltime based on the input run. This variation differs from task to task. For instance, task #7 has variations up to 25 minutes while tasks #3 and #4 have less than 5 minutes difference between runs.




(a) Typical memory profile with OASIS input.




(b) Typical memory profile with DRD input.

Figure 5.5: Examples of memory footprints of the SLANT application with inputs from each considered dataset. Memory consumption is measured every 2 seconds with the *used memory* field of the `vmstat` command.

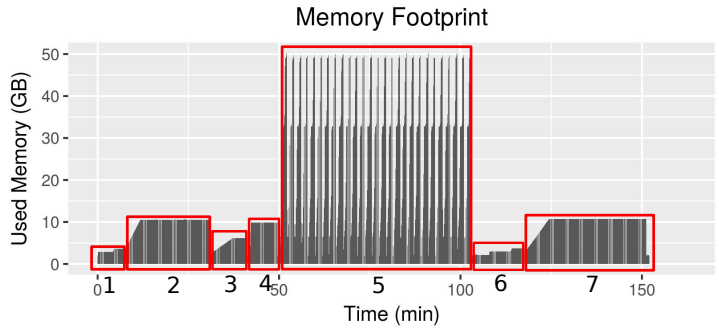



Figure 5.6: Job decomposition in tasks based on raw data of a memory footprint.

Another observation from the raw data on Figure 5.7, is that some tasks present several peaks (tasks #5 and #7). There may be several explanations to this, from actual task profile (for instance a condition that adds a lot of work if it is met), lack of sufficient data for a complete profile, or finally a bad choice in our task decomposition. Going further, one may be interested in generating a finer grain parsing of the application profile to separate these peaks into individual tasks, based on more parameters than only the memory consumption. We choose not to do this to preserve some simplicity to our model. In the following, we denote by $X_1, \ldots, X_7$ the random variable that represents the execution times of the seven tasks.

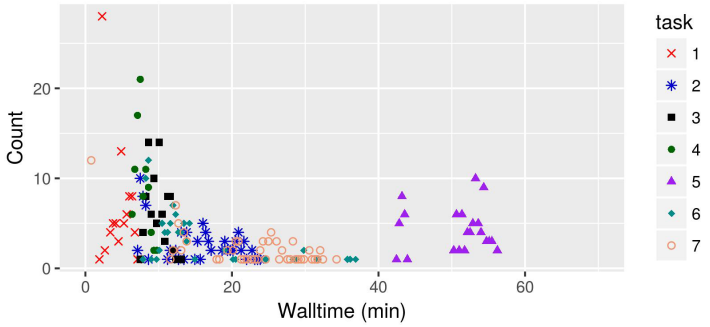



Figure 5.7: Count of the task walltime for all jobs (raw data).

Table 5.1: Pearson Correlation matrix of the walltimes of the different tasks.

| Task Index | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 1.000 | 0.998 | -0.308 | -0.261 | -0.114 | -0.039 | 0.139 |
| 2 | | 1.000 | -0.293 | -0.277 | 0.142 | -0.058 | 0.159 |
| 3 | | | 1.000 | 0.076 | 0.547 | -0.283 | 0.223 |
| 4 | | | | 1.000 | -0.361 | 0.296 | -0.308 |
| 5 | | | | | 1.000 | -0.568 | 0.574 |
| 6 | | | | | | 1.000 | -0.475 |
| 7 | | | | | | | 1.000 |

An important next question is whether they show correlation in their variation. Indeed, given that they are based on the same input, one may assume that they vary similarly. To study this, we present in Table 5.1 their Pearson Correlation coefficients. We see that only tasks #1 and #2 present a very high correlation (meaning that their execution times are proportional), while others have meaningless correlation. This measure is important as it hints at the independence of the different execution time variables.

Finally, to investigate the distribution of memory usage over time, we study the task status at all time (at time $t$, which task is being executed). To do so, given $X_i$ ($i = 1 \dots 7$) the execution time of task $i$, we represent in Figure 5.8 the functions $y_i(t) = \mathbb{P}\left(\sum_{j \leq i} X_j < t\right)$. Essentially, it means that $y_i$ is the probability that task $i$ is finished.

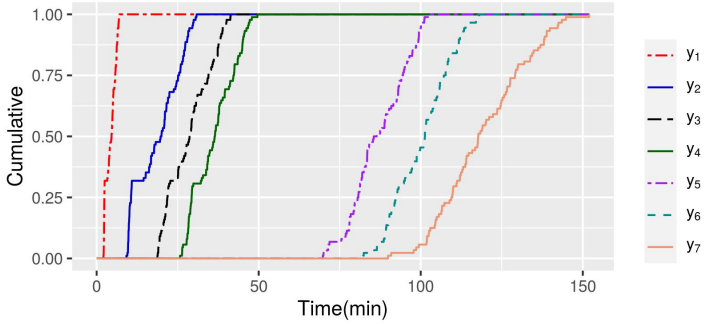



Figure 5.8: $y_i(t) = \mathbb{P}\left(\sum_{j \leq i} X_j < t\right)$ is the probability that task $i$ is finished at time $t$ (raw data).

Figure 5.8 is read this way: the probability that task $i$ is running at time $t$ corresponds to the distance between the plots corresponding to task $i-1$ and task $i$. For instance, at time $t=0$ task #1 is running with probability 1. At time 100, tasks #5 to #7 are running (roughly) with respective probability 0.06, 0.5, 0.38. In addition, with probability 0.06 the job has finished its execution.

This figure is interesting in the sense that it gives task properties as a function of time. For instance, given the memory footprint of each task, one can estimate the probability of the different memory needs.

## 5.3 From observations to a theoretical model

Using the observations from Section 5.2.1, we now derive a new computational model. We discuss the advantages and limitations of this model in Section 5.3.2.

### 5.3.1 Job model

We model an application $A$ as a chain of $n$ tasks:

$$A = j_1 \to j_2 \to \cdots \to j_n,$$

such that $j_i$ cannot be executed until $j_{i-1}$ is finished. Each task $j_i$ is defined by two parameters: an execution time and a peak memory footprint. The peak memory footprint of each task does not depend on the input, and hence can be written as $M_i$. The execution time of each task is however input dependent, and we denote by $X_i$ the random variable that represents the execution time of task $j_i$. $X_i$ follows a probability distribution of density (PDF) $f_i$. We also assume that the $X_i$ are independent.

Finally, the compact way to represent an application is

$$\{(f_1, M_1), \ldots, (f_n, M_n)\}. \tag{5.1}$$

### 5.3.2 Discussion

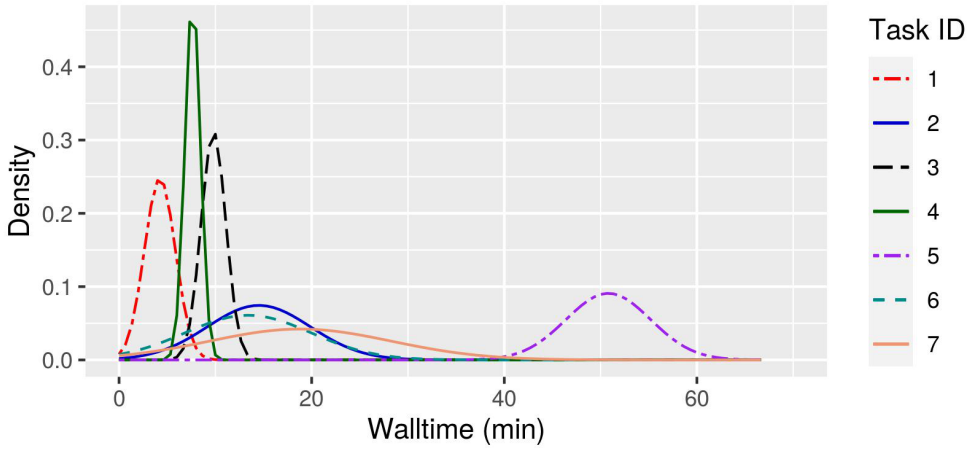



Figure 5.9: Interpolation of data from Figure 5.7 with Normal Distributions.

To discuss the model, we propose to interpolate the data from our application with Normal Distributions[4]. We present such an interpolation on Figure 5.9 (data in Table 5.2). Fitting to continuous

[4]We write that $X$ follows a normal distribution $\mathcal{N}(\mu, \sigma)$.

Table 5.2: Parameters $(\mu, \sigma)$ of the Normal Distributions interpolated in Figure 5.9.

| Task ID | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Mean $\mu$ (in sec) | 255 | 871 | 588 | 459 | 3050 | 804 | 1130 |
| Std $\sigma$ (in sec) | 96.7 | 322 | 76.8 | 48.1 | 263 | 393 | 568 |

distributions is interesting in terms of data representation, and offers more flexibility to study the properties of the application. As we have seen earlier, Normal Distributions may not be the best candidate for those jobs (for examples jobs with multiple peaks), but they have the advantage of being simpler to manipulate. This is also a good element to discuss the limitations of our model.

Using the interpolations, one can then compute several quantities related to the problem with more or less precision. We show how one would proceed in the following.

### Task status with respect to time

We can estimate the functions $\mathbb{P}\left(\sum_{j\leq i} X_j < t\right)$ represented in Figure 5.8, which later helps to guess the task status with respect to time. Indeed, if $X_1, \ldots, X_i$ are independent normal distributions of parameters $\mathcal{N}(\mu_1, \sigma_1), \ldots, \mathcal{N}(\mu_i, \sigma_i)$, then $Y_i = \sum_{j\leq i} X_j$ follows $\mathcal{N}(\sum_{j\leq i} \mu_j, \sqrt{\sum_{j\leq i} \sigma_j^2})$. We plot in Figure 5.10 the functions $f_i = \mathbb{P}(Y_i < t)$.

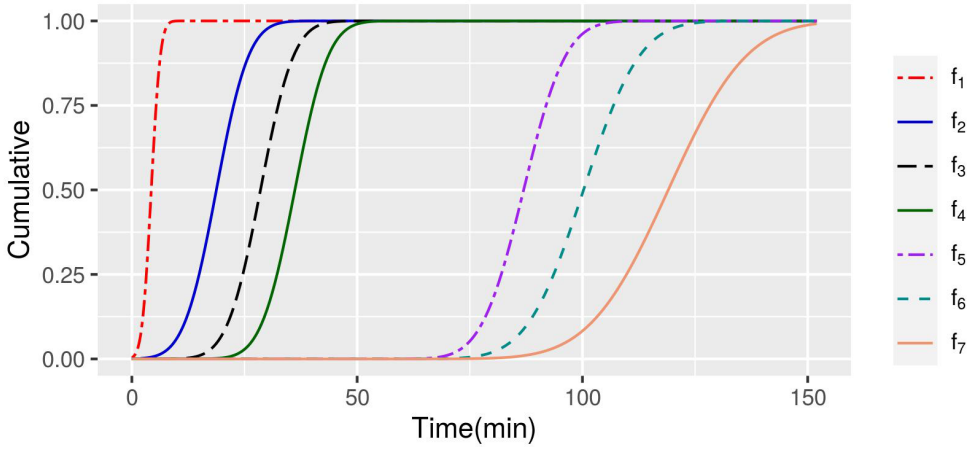



Figure 5.10: Representation of the cumulative distribution of the termination time of the 7 tasks over time from raw data.

An important observation from this figure is that even if the interpolations per task are not perfect, the sum of their model gets *closer* with time to actual data. Obviously this may not be true for all applications and is subject to caution, however the fact that initially all models seemed far off on a per task basis but converged well is positive.

### Memory specific quantities

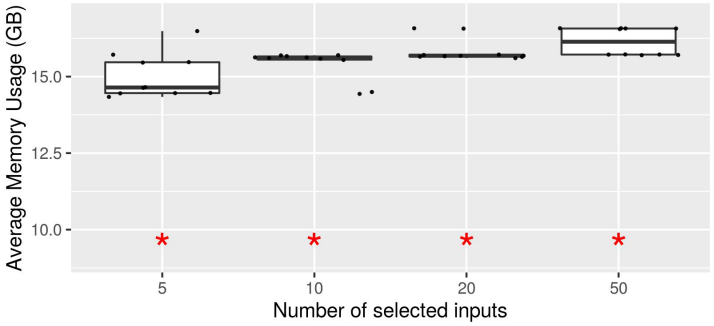
Using this data, one should be able to compute different grandeurs needed for an evaluation, such as:

- The average memory needed for a run $\bar{M} = \sum_{i=1}^{n} M_i \mathbb{E}[X_i] / \sum_{i=1}^{n} \mathbb{E}[X_i]$. This quantity may be useful for co-scheduling schemes in the case of shared/overprovisionned resources [11, 59];
- Or even arbitrary values such as, the "likely" maximum memory needed as a function of time.

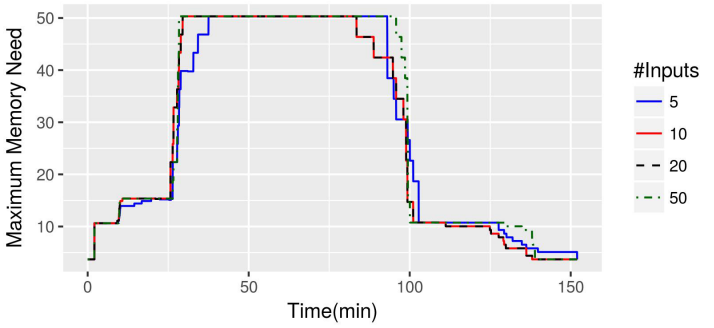
$$M_\tau(t) = \max \left\{ M_i | \mathbb{P} \left( \sum_{j<i} X_j < t \leq \sum_{j \leq i} X_j \right) > \tau \right\} \tag{5.2}$$

In addition, the data for the values of $M_i$ can be obtained with traces of very few executions (since it is not input dependent).

The $f_i$ can also be interpolated from very few executions with more or less precision. We evaluate this precision here with the following experiment, presented in Figure 5.11. We interpolate from 5, 10, 20, 50 randomly selected (with replacement) runs the functions $f_i$ and compare (i) the evolution of $\bar{M}$; and (ii) the maximum memory need $t \mapsto M_{0.1}(t)$. Each experiment is repeated 10 times to study the variations.




(a) Average memory $\bar{M}$ for different number of inputs over 10 experiments. Red star is $\bar{M}$ of the original 88 runs.




(b) $M_{0.1}$ for different number of inputs (avg of 10 experiments).

Figure 5.11: The model can help interpolate different quantities such as average memory (top) or peak memory (bottom).

We observe from Figure 5.11a that with respect to the average memory need, increasing the number of data elements does not improve the precision significantly. This was expected since the only information needed is the expectation of the random variables, which is a lot easier to obtain than the distribution. The difference between $\bar{M}$ as evaluated and the red star is due to the job modelisation. In the model, we consider constant memory per task when it is not the case. For instance the memory of

Task 5 is set at 50GB in the model (and in the computation of $\bar{M}$), when in practice (Fig. 5.6) it fluctuates a lot, hence the red star being lower.

With respect to the maximum memory requirements (Figure 5.11b), it seems that very few runs (5 runs) already give good performance. This could also be predicted due to the *Maximum* function which gives more weight to any single run.

Obviously this modelization is not perfect and can be improved depending on the level of precision one needs, specifically we can see the following caveats:

- The peak memory is different from the average memory usage (see for instance task #5 in Figure 5.6), where the job varies between high-memory needs and low-memory needs. Hence using peak memory to guess the average memory may lead to an overestimation of the average memory (as shown in Figure 5.11a). To mitigate this, one may add as a variable the average memory per task.

- The model assumes that the lengths of the tasks are independent. However this may not be true as we have seen in Table 5.1 where the lengths of tasks #1 and #2 are highly correlated. In our case, a simple way to fix this would have been to merge them into a single meta task. We chose not to do this to study the limits of the model.

- This model is based on the information available today. Specifically, the jobs here are sequentialized (the dependencies are represented by a chain of tasks). However we can expect a more general formulation where the dependencies are more parallel (and hence represented by a Directed Acyclic Graph instead of a linear chain).

To conclude this section, we have presented a model for the novel HPC applications that is easy to manipulate but still seems close to the actual performance. We discussed possible limitations to this model.

## 5.4 Model instantiation and performance

Question 3.1 incorporated a notion of *expected quality of the information obtainable*. Given this model and given the expected quality of the information obtainable, are we able to design algorithmic strategy that perform well compared to what we are evaluating?

In our work [J5], and given the observations, we proposed to interpolate the distribution based on historical data. How many historical points do we need for performance that are good enough? Note that few points may give an interpolation that is inaccurate, but it does not mean that the performance of the algorithm is bad (remember that this is the difference between Question 1.1 and Question 1.2).

**Example.** With Ana Gainaru, Brice Goglin and Valentin Honoré [J5] we compared various algorithms to compute the reservation strategies. All these strategies are based from the same input: $k$ previous runs of the application (in practice we use $k = 5,\ 10,\ 20,\ 50$).

- ALL-CKPT [C6, III.D]: This computes the optimal solution to minimize the expected total reservation time when all reservations are checkpointed and when the checkpoint cost is constant. We take the maximum memory footprint over the execution as the basis for the checkpoint cost.

- MEM-ALL-CKPT: it is an extension of ALL-CKPT based on Section 5.3.1. Specifically it

uses $M_{0.1}$ (defined in Eq. (5.2)) as the basis for the checkpoint cost function. The complete procedure of this extension can be found in [J5].

- NEURO [J9, 44]: This is the algorithm used by the neuroscience department at Vanderbilt University. In their algorithm, they use the maximum length of the last $k$ runs as their first reservation. If it is not enough they multiply it by 1.5 and repeat the procedure. To be fair with the other strategies, we added a checkpoint to this strategy. Hence the length of the second reservation ($T_2$) is only 50% of the first one ($T_1$), so that $T_1 + T_2 = 1.5T_1$. We use the maximum size of a checkpoint as checkpoint cost. For completeness, we have also added a strategy that uses average length instead of maximum length. We denote it by NEURO-AVG.

Overall and without going into too much details here, we were able to show that thanks to checkpointing, the solution is extremely robust: using only the five last runs, one could obtain a performance almost identical to the one with fifty runs. Without checkpointing [C10], the convergence is slower, but we could obtain interesting performance for many job distributions.

## 5.5 General comments

I have described here the experimental construction of a new job model that could be better fitted for the representation of certain jobs. The underlying hypothesis is that the information of job performance is inaccurate because the data is non deterministic. This model did make purposefully some wrong asumption (for instance the independence between the length of several tasks).

Of course, the model proposed now needs to be evaluated on a lot more jobs, this is a direction that I am working on is to study statistical behavior model / non-deterministic models, wondering if this could be more correct. These models may be much harder to manipulate algorithmically, but they may be more accurate and provide better expected performance.

# Conclusion

In this habilitation thesis I have revisited how one could design models for resource management in HPC. I have proposed and studied several design hypotheses. I have discussed and illustrated the fact that having a model that is *too* accurate may be counter-productive: it complicates the algorithm design without necessarily improving the final performance.

Going further, I have tried to demonstrate that models should start to take into account the fact that the input information may be inaccurate. This is I believe an important paradigm shift for scheduling in HPC resource management, where previous studies have mostly focused on trying to get accurate data. Then, I discussed the fact that the optimization objectives should also be reevaluated: by focusing on optimizing a single objective, there is a risk of losing sight of what is actually happening in terms of schedule.

Overall, I believe that this document presents an overview of how my research vision evolved in the last 10 years; interestingly I believe that this vision has been highly influenced (for the better) by my involvement in the *Inria Evaluation Committee*[5]: indeed, as part of this committee, I have worked on a report on how to evaluate research in general and the risks of focusing on specific indicators [O2]. I believe the same type of risks apply to our field.

## Future of the field and political considerations

In the rest of this conclusion I would like to discuss several elements of vision that I have on the evolution of our field, in relation to the ecological crisis that we are living through.

When I ask myself what the future of HPC may look like[6], there are several elements that I would like to consider in my future research:

**Chip shortage** Chip shortage is likely to occur for many reason: many component of these chips are mostly produced in a single country (for instance for Rare Earth Element, China)[7]. These are called Critical raw materials when not only are they important in our daily use, but when this importance is combined to a high risk associated to their supply. Indeed production could stop for other part of the world due to a socio-political crisis. In addition, they are extremely costly from an environmental [26] and a human perspective to produce [49, 16, 41]. Finally, we have also seen that extreme weather changes may impact the production of chips, for instance recently (Summer 2021), droughts in Taiwan affected chip production as they impacted the availability of ultra-pure water that is needed to clean the silicon wafers used in microprocessors[8].

[5] This committee is in charge of all type of scientific evaluation at Inria (including prospective work, recruitment, team evaluation, advancement etc.)

[6] Under the hypothesis that we still have some sort of HPC, hypothesis that is far from being granted.

[7] Critical Raw Materials Resilience: Charting a Path towards greater Security and Sustainability, European Commission, 2020

[8] Taiwan's drought is exposing just how much water chipmakers like TSMC use (and reuse), Eamon Barrett, Fortune, 2021

All in all, shortage of such critical raw material is something that is likely to occur in a close future. Will we be able to build a new supercomputer in 2030? in 2050? What would HPC look like in the case where we are not able to renew our machines every 5 years? There is a chance that this is not the end of HPC, but the beginning of a different HPC. We may observe new scheduling constraints for instance with the introduction of second-hand equipment:

1. if we start using *old* architecture, then we may see an increasing number of failures. Currently the research in fault tolerance makes the assumption that the mean time between failure is large before the checkpoint time. Without this hypothesis, checkpointing does not work anymore, hence reinventing resilience strategies may be critical.

2. Some part of the architecture may even be definitely *broken*, or with variable performance: how do we include this heterogeneity/variable performance into our scheduling models? This is again an information that may be extremely inaccurate.

More generally, even if this shortage does not occur at short-term, is it really sustainable to have HPC machines with a lifespan of 5 years? I believe those questions will gain tremendous important in a close future.

**Energy shortage or Renewable energy** The problematic of having computing centers whose sole energy supply is renewable energy is also getting increasing attention. One of the *novelty*[9] of renewable energy is its intermittence, the fact that the source of energy is not constant but may be extremely low at some times (for instance on cloudy days). There has been an increasing amount of research on this topic. Particularly, it was discussed in a new working group on variable capacity resources for sustainability. In the panel *Unspoken Challenges* at this workshop, I was able to expose my vision of a key problematic that I believe we face: the race for performance.

Indeed, I believe that the variability of availability of resource should reopen for us some assumptions that we made about HPC systems, amongst which:

1. Time criticality: HPC has always been evaluated on performance. How to get things fast; Which machine has the best response time etc. I believe that we need to start thinking about priority-based HPC centers where not all jobs have the same priority, or *Not-Urgent Computing*. Managing jobs in an HPC center has often been about *fairness* (which does not seem absurd). The First-Come-First-Served heuristics guarantees that no job will be delayed too much. But could we be more efficient from an energetic perspective if we did not impose this constraint and informed all users that there is in general a one-week response time and this is not negotiable.

2. "Meta"-research: how to evaluate performance. As we have seen in chapter 3, when designing a solution to optimize a criteria, this may have unexpected (and possibly detrimental) effects. For instance, minimizing the mean bounded slowdown via Reinforcement Learning is likely to prioritize the smallest jobs instead of the large parallel jobs as one may “expect” on a supercomputer.

   Are scheduling models as we use to do them still the correct way to evaluate performance? This is also the discussion that I wanted to raise with this habilitation thesis For instance when looking at a single optimization objective, there is a high risk of rebound effect. Hence we really need a qualitative assessment of our solutions instead of numerical optimization objectives (i.e. I do not believe that "minimizing the energy consumption for X" will suffice to solve the energy consumption problem).

[9] From a scheduling perspective

How users react should be an extremely important part of our research, and needs to be done in coordination with social science researchers.

3. Reviewing best practices: *"What happens if I try this configuration? It doesn't work, what about this one?" "Let me collect all possible data and think later what I am going to do with them." "Now that I am doing my evaluation I am realizing that I could use this information, let me re-run this experiment"*. These are all statements that I have seen during my years as a researcher.

   Generally, I am under the impression that contrarily to other experimental sciences where the cost of setting up an experimentation is extremely visible (think about a chemistry experiment, where 2ml of a reactant cost more than $1000), computing-based science has often lacked the vision of correctly setting up an experiment. The cost of computation is often decorrelated from our research. Some large computing centers ask users to request a number of core-hour for an experiment, but in my experience it is often largely before an evaluation protocol is designed, and the value is often over-estimated.

   Interestingly, during the panel discussed earlier, I simply brushed the idea that we could be getting reports of the computing cost of our evaluation, and including them into our papers (as is already done by some communities or some authors [18]). This was a panel in a small workshop with people environmentally-concerned. Yet the opposition to this idea was extremely visible

**Final words** To conclude, I believe that there are an important number of challenges revolving around HPC and the climate catastrophe. HPC scientists cannot simply be spectators to this, and we really need to rethink about how our system works. I also believe that for us, it goes further than simply doing resource management as usual with a "minimizing the energy consumption" criterion. This is the direction that I would like to give to my future research.

# Bibliography


[1] "Slurm Workload Manager," https://slurm.schedmd.com/, version 20.02. Accessed: 2020-10-06.

[2] J. Abella, D. Hardy, I. Puaut, E. Quinones, and F. J. Cazorla, "On the comparison of deterministic and probabilistic wcet estimation techniques," in *2014 26th Euromicro Conference on Real-Time Systems*. IEEE, 2014, pp. 266–275.

[3] S. Ali, H. J. Siegel, and A. A. Maciejewski, "Perspectives on robust resource allocation for heterogeneous parallel and distributed systems," in *Handbook of Parallel Computing Models, Algorithms and Applications*. MK Publishing, 2008.

[4] S. Altmeyer, L. Cucu-Grosjean, and R. I. Davis, "Static probabilistic timing analysis for real-time systems using random replacement caches," *Real-Time Systems*, vol. 51, no. 1, pp. 77–123, 2015.

[5] S. Altmeyer and R. I. Davis, "On the correctness, optimality and precision of static probabilistic timing analysis," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2014, pp. 1–6.

[6] L. Bautista-Gomez, A. Gainaru, S. Perarnau, D. Tiwari, S. Gupta, C. Engelmann, F. Cappello, and M. Snir, "Reducing waste in extreme scale systems through introspective analysis," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2016, pp. 212–221.

[7] L. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, N. Maruyama, and S. Matsuoka, "Fti: High performance fault tolerance interface for hybrid systems," in *Proceedings of 2011 international conference for high performance computing, networking, storage and analysis*, 2011, pp. 1–32.

[8] D. Bertsimas and M. Sim, "Robust discrete optimization and network flows," *Mathematical programming*, vol. 98, no. 1, pp. 49–71, 2003.

[9] J. L. Bez, F. Z. Boito, R. Nou, A. Miranda, T. Cortes, and P. O. Navaux, "Adaptive request scheduling for the i/o forwarding layer using reinforcement learning," *Future Generation Computer Systems*, vol. 112, pp. 1156–1169, 2020.

[10] M. Bougeret, A. A. Pessoa, and M. Poss, "Robust scheduling with budgeted uncertainty," *Discrete Applied Mathematics*, vol. 261, pp. 93–107, 2019.

[11] J. Breitbart, S. Pickartz, S. Lankes, J. Weidendorfer, and A. Monti, "Dynamic co-scheduling driven by main memory bandwidth utilization," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, 2017, pp. 400–409.

[12] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst, "hwloc: A generic framework for managing hardware affinities in hpc applications," in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2010, pp. 180–186.

[13] J. Bruno, P. Downey, and G. N. Frederickson, "Sequencing tasks with exponential service times to minimize the expected flow time or makespan," *Journal of the ACM (JACM)*, vol. 28, no. 1, pp. 100–113, 1981.

[14] D. Carastan-Santos and R. Y. De Camargo, "Obtaining dynamic scheduling policies with simulation and machine learning," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–13.

[15] D. Carastan-Santos, R. Y. De Camargo, D. Trystram, and S. Zrigui, "One can only gain by replacing easy backfilling: A simple scheduling policies case study," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2019, pp. 1–10.

[16] B. Cheruga, R. Liron, and M. Canavera, "Ensuring children's social protection in the democratic republic of the congo: A case study of combating child labour in the copper-cobalt belt," *What works for Africa's poorest children: From measurement to action*, p. 273, 2020.

[17] W. Cirne and F. Berman, "A comprehensive model of the supercomputer workload," in *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No. 01EX538)*. IEEE, 2001, pp. 140–148.

[18] T. Cornebize, "High Performance Computing : towards better Performance Predictions and Experiments," Theses, Université Grenoble Alpes [2020-....], June 2021. [Online]. Available: https://theses.hal.science/tel-03328956

[19] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiñones, and F. J. Cazorla, "Measurement-based probabilistic timing analysis for multi-path programs," in *2012 24th euromicro conference on real-time systems*. IEEE, 2012, pp. 91–101.

[20] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," vol. 22, no. 3. Elsevier, 2006, pp. 303–312.

[21] R. L. Daniels and P. Kouvelis, "Robust scheduling to hedge against processing time uncertainty in single-stage production," *Management Science*, vol. 41, no. 2, pp. 363–376, 1995.

[22] R. I. Davis and L. Cucu-Grosjean, "A survey of probabilistic timing analysis techniques for real-time systems," *Leibniz Trans. Embed. Syst.*, vol. 6, no. 1, pp. 03:1–03:60, 2019.

[23] Y. Fan, Z. Lan, T. Childers, P. Rich, W. Allcock, and M. E. Papka, "Deep reinforcement agent for scheduling in hpc," 2021.

[24] Y. Fan, P. Rich, W. E. Allcock, M. E. Papka, and Z. Lan, "Trade-off between prediction accuracy and underestimation rate in job runtime estimates," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2017, pp. 530–540.

[25] I. Fridman, E. Pesch, and Y. Shafransky, "Minimizing maximum cost for a single machine under uncertainty of processing times," *European Journal of Operational Research*, vol. 286, no. 2, pp. 444–457, 2020.

[26] S. R. Golroudbary, I. Makarava, A. Kraslawski, and E. Repo, "Global environmental cost of using rare earth elements in green energy technologies," *Science of The Total Environment*, vol. 832, p. 155022, 2022.

[27] A. V. Goponenko, K. Lamar, C. Peterson, B. A. Allan, J. M. Brandt, and D. Dechev, "Metrics for packing efficiency and fairness of hpc cluster batch job scheduling," in *2022 IEEE 34th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2022, pp. 241–252.

[28] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," in *Annals of discrete mathematics*. Elsevier, 1979, vol. 5, pp. 287–326.

[29] N. Grinsztajn, O. Beaumont, E. Jeannot, and P. Preux, "Geometric deep reinforcement learning for dynamic dag scheduling," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2020, pp. 258–265.

[30] S. Gupta, T. Patel, C. Engelmann, and D. Tiwari, "Failures in large scale systems: long-term measurement, analysis, and implications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.

[31] T. Hansen, F. M. Ciorba, A. A. Maciejewski, H. J. Siegel, S. Srivastava, and I. Banicescu, "Heuristics for robust allocation of resources to parallel applications with uncertain execution times in heterogeneous systems with uncertain availability," in *Proceedings of the World Congress on Engineering*, vol. 1, 2014.

[32] J. Haxby, J. S. Guntupalli, A. Connolly, Y. Halchenko, B. Conroy, M. Gobbini, M. Hanke, and P. Ramadge, "A common, high-dimensional model of the representational space in human ventral temporal cortex," *Neuron*, vol. 72, pp. 404–16, 10 2011.

[33] E. Heien, D. Kondo, A. Gainaru, D. LaPine, B. Kramer, and F. Cappello, "Modeling and tolerating heterogeneous failures in large parallel systems," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*. IEEE, 2011, pp. 1–11.

[34] T. Herault and Y. Robert, *Fault-tolerance techniques for high-performance computing*. Springer, 2015.

[35] Y. Huo, Z. Xu, K. Aboud, P. Parvathaneni, S. Bao, C. Bermudez, S. M. Resnick, L. E. Cutting, and B. A. Landman, "Spatially localized atlas network tiles enables 3d whole brain segmentation from limited data," in *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018*, A. F. Frangi, J. A. Schnabel, C. Davatzikos, C. Alberola-López, and G. Fichtinger, Eds. Cham: Springer International Publishing, 2018, pp. 698–705.

[36] Y. Huo, Z. Xu, Y. Xiong, K. Aboud, P. Parvathaneni, S. Bao, C. Bermudez, S. M. Resnick, L. E. Cutting, and B. A. Landman, "3d whole brain segmentation using spatially localized atlas network tiles," *NeuroImage*, vol. 194, pp. 105 – 119, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1053811919302307

[37] IOR Benchmark, "version 3.3.0," https://github.com/hpc/ior, 2021.

[38] M. Isakov, E. Del Rosario, S. Madireddy, P. Balaprakash, P. Carns, R. B. Ross, and M. A. Kinsy, "Hpc i/o throughput bottleneck analysis with explainable local models," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–13.

[39] S. Islam, B. Bode, and J. Enos, "Blue waters resource management and job scheduling best practices," 2016.

[40] A. Kasperski and P. Zieliński, "Single machine scheduling problems with uncertain parameters and the owa criterion," *J. of Scheduling*, vol. 19, no. 2, p. 177–190, April 2016.

[41] A. Kelly, "Apple and google named in us lawsuit over congolese child cobalt mining deaths," *The Guardian*, vol. 16, 2019.

[42] D. Kondo, B. Javadi, A. Iosup, and D. Epema, "The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems," *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, vol. 0, pp. 398–407, 2010.

[43] P. J. LaMontagne, T. L. Benzinger, J. C. Morris, S. Keefe, R. Hornbeck, C. Xiong, E. Grant, J. Hassenstab, K. Moulder, A. Vlassenko, M. E. Raichle, C. Cruchaga, and D. Marcus, "Oasis-3: Longitudinal neuroimaging, clinical, and cognitive dataset for normal aging and alzheimer disease," *medRxiv*, 2019. [Online]. Available: https://www.medrxiv.org/content/early/2019/12/15/2019.12.13.19014902

[44] B. Landman, "Medical-image Analysis and Statistical Interpretation (MASI) Lab," https://my.vanderbilt.edu/masi/.

[45] LANL, "Computer failure data repository," https://www.usenix.org/cfdr-data, 2006. [Online]. Available: https://www.usenix.org/cfdr-data

[46] S. LANL, NERSC, "Apex workflows," https://www.nersc.gov/assets/apex-workflows-v2.pdf, Technical report, LANL, NERSC, SNL, Tech. Rep., 2016.

[47] C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely, "Are user runtime estimates inherently inaccurate?" in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2004, pp. 253–263.

[48] A. Legrand, D. Trystram, and S. Zrigui, "Adapting batch scheduling to workload characteristics: What can we expect from online learning?" in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2019, pp. 686–695.

[49] K. Li, T. Liang, L. Wang, and Z. Yang, "Contamination and health risk assessment of heavy metals in road dust in bayan obo mining region in inner mongolia, north china," *Journal of Geographical Sciences*, vol. 25, pp. 1439–1451, 2015.

[50] C. Liebchen, M. Lübbecke, R. Möhring, and S. Stiller, "The concept of recoverable robustness, linear programming recovery, and railway applications," in *Robust and online large-scale optimization*. Springer, 2009, pp. 1–27.

[51] C.-D. Lu, "Failure data analysis of hpc systems," *arXiv preprint arXiv:1302.4779*, 2013.

[52] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[53] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, ser. HotNets '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 50–56. [Online]. Available: https://doi.org/10.1145/3005745.3005750

[54] M. Mastrolilli, N. Mutsanas, and O. Svensson, "Approximating single machine scheduling with scenarios," in *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*. Springer, 2008, pp. 153–164.

[55] E. Mocanu, D. C. Mocanu, P. H. Nguyen, A. Liotta, M. E. Webber, M. Gibescu, and J. G. Slootweg, "On-line building energy optimization using deep reinforcement learning," *IEEE transactions on smart grid*, vol. 10, no. 4, pp. 3698–3708, 2018.

[56] A. W. Mu'alem and D. G. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling," *IEEE transactions on parallel and distributed systems*, vol. 12, no. 6, pp. 529–543, 2001.

[57] J. Niño-Mora, *Stochastic Scheduling*. Boston, MA: Springer US, 2009, pp. 3818–3824.

[58] T. Patel, Z. Liu, R. Kettimuthu, P. Rich, W. Allcock, and D. Tiwari, "Job characteristics on large-scale systems: Long-term analysis, quantification and implications," in *2020 SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE Computer Society, 2020, pp. 1186–1202.

[59] T. Patki, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. De Supinski, "Exploring hardware overprovisioning in power-constrained, high performance computing," in *Proceedings of the 27th international ACM conference on International conference on supercomputing*, 2013, pp. 173–182.

[60] D. Petcu, "The performance of parallel iterative solvers," *Computers and Mathematics with Applications*, vol. 50, no. 7, pp. 1179–1189, 2005.

[61] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Society for Industrial and Applied Mathematics, 2003.

[62] L. Santinelli, F. Guet, and J. Morio, "Revising measurement-based probabilistic timing analysis," in *2017 IEEE real-time and embedded technology and applications symposium (RTAS)*. IEEE, 2017, pp. 199–208.

[63] K. P. Silva, L. F. Arcaro, and R. S. De Oliveira, "On using gev or gumbel models when applying evt for probabilistic wcet estimation," in *2017 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2017, pp. 220–230.

[64] M. Skutella and M. Uetz, "Stochastic machine scheduling with precedence constraints," *SIAM Journal on Computing*, vol. 34, no. 4, pp. 788–802, 2005.

[65] S. Snyder, P. Carns, K. Harms, R. Ross, G. K. Lockwood, and N. J. Wright, "Modular HPC I/O characterization with darshan," in *2016 5th workshop on extreme-scale programming tools (ESPT)*. IEEE, 2016, pp. 9–17.

[66] A. Souza, K. Pelckmans, D. Ghoshal, L. Ramakrishnan, and J. Tordsson, "Asa-the adaptive scheduling architecture," in *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, 2020, pp. 161–165.

[67] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova, "Resource allocation algorithms for virtualized service hosting platforms," *Journal of Parallel and Distributed Computing*, vol. 70, no. 9, pp. 962–974, May 2010. [Online]. Available: https://hal.inria.fr/inria-00527066

[68] H. Sung, J. Bang, C. Kim, H.-S. Kim, A. Sim, G. K. Lockwood, and H. Eom, "BBOS: Efficient HPC Storage Management via Burst Buffer Over-Subscription," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, 2020, pp. 142–151.

[69] M. Tanash, B. Dunn, D. Andresen, W. Hsu, H. Yang, and A. Okanlawon, "Improving hpc system performance by predicting job resources via supervised machine learning," in *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning)*, ser. PEARC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3332186.3333041

[70] W. Tang, N. Desai, D. Buettner, and Z. Lan, "Analyzing and adjusting user runtime estimates to improve job scheduling on the blue gene/p," in *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE, 2010, pp. 1–11.

[71] W. Tang, Z. Lan, N. Desai, and D. Buettner, "Fault-aware, utility-based job scheduling on blue, gene/p systems," in *2009 IEEE International Conference on Cluster Computing and Workshops*. IEEE, 2009, pp. 1–10.

[72] A. J. S. Tipu, P. Ó. Conbhuí, and E. Howley, "Applying neural networks to predict hpc-i/o bandwidth over seismic data on lustre file system for exseisdat," *Cluster Computing*, vol. 25, no. 4, pp. 2661–2682, 2022.

[73] M. Tirmazi, A. Barker, N. Deng, M. E. Haque, Z. G. Qin, S. Hand, M. Harchol-Balter, and J. Wilkes, "Borg: The next generation," in *Proceedings of the Fifteenth European Conference on Computer Systems*, ser. EuroSys '20. New York, NY, USA: Association for Computing Machinery, 2020.

[74] D. Tiwari, S. Gupta, and S. S. Vazhkudai, "Lazy checkpointing: Exploiting temporal locality in failures to mitigate checkpointing overheads on extreme-scale systems," in *44th Int. Conf. Dependable Systems and Networks*. IEEE, 2014, pp. 25–36.

[75] S. Toueg and O. Babaoğlu, "On the optimum checkpoint selection problem," *SIAM J. Comput.*, vol. 13, no. 3, 1984.

[76] D. Tsafrir, "Using inaccurate estimates accurately," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2010, pp. 208–221.

[77] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling using system-generated predictions rather than user runtime estimates," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 789–803, 2007.

[78] Tsubame, "Failure history," http://mon.g.gsic.titech.ac.jp/trouble-list/index.htm, 2017. [Online]. Available: http://mon.g.gsic.titech.ac.jp/trouble-list/index.htm

[79] P. Velho, L. M. Schnorr, H. Casanova, and A. Legrand, "On the validity of flow-level tcp network models for grid and cloud simulations," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 23, no. 4, pp. 1–26, 2013.

[80] R. R. Weber, "Scheduling jobs with stochastic processing requirements on parallel machines to minimize makespan or flowtime," *Journal of Applied Probability*, pp. 167–182, 1982.

[81] G. Weiss, "Turnpike optimality of smith's rule in parallel machines stochastic scheduling," *Mathematics of Operations Research*, vol. 17, no. 2, pp. 255–270, 1992.

[82] M. R. Wyatt, S. Herbein, T. Gamblin, A. Moody, D. H. Ahn, and M. Taufer, "Prionn: predicting runtime and io using neural networks," in *Proceedings of the 47th International Conference on Parallel Processing*, 2018, pp. 1–12.

[83] W. Yang, X. Liao, D. Dong, and J. Yu, "A quantitative study of the spatiotemporal i/o burstiness of hpc application," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2022, pp. 1349–1359.

[84] J. W. Young, "A first order approximation to the optimum checkpoint interval," vol. 17, no. 9, 1974, pp. 530–531.

[85] D. Zhang, D. Dai, Y. He, F. S. Bao, and B. Xie, "Rlscheduler: an automated hpc batch job scheduler using reinforcement learning," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–15.

[86] D. Zhang, D. Dai, and B. Xie, "Schedinspector: A batch job scheduling inspector using reinforcement learning," 2022.

[87] X. Zhang, K. Davis, and S. Jiang, "IOrchestrator: Improving the performance of multi-node I/O systems via inter-server coordination," in *SC'10*. IEEE, 2010, pp. 1–11.

[88] Z. Zhou, X. Yang, D. Zhao, P. Rich, W. Tang, J. Wang, and Z. Lan, "I/O-aware batch scheduling for petascale computing systems," in *2015 IEEE International Conference on Cluster Computing*. IEEE, 2015, pp. 254–263.

# Publications

## Articles in international refereed journals


[J1] Yiqin Gao, Guillaume Pallez, Yves Robert, and Frédéric Vivien. Dynamic scheduling strategies for firm semi-periodic real-time tasks. *IEEE Transactions on Computers*, 1(1):55–68, 2023.

[J2] Yishu Du, Loris Marchal, Guillaume Pallez, and Yves Robert. Optimal checkpointing strategies for iterative applications. *IEEE Trans. Parallel Distributed Syst.*, 33(3):507–522, 2022.

[J3] Guillaume Aupy, Emmanuel Jeannot, and Nicolas Vidal. Scheduling periodic i/o access with bi-colored chains: models and algorithms. *J. Scheduling*, 2021.

[J4] Olivier Beaumont, Julien Herrmann, Guillaume Pallez, and Alena Shilova. Optimal memory-aware backpropagation of deep join networks. *Philosophical Transactions of the Royal Society A*, 378(2166):20190049, 2020.

[J5] Ana Gainaru, Brice Goglin, Valentin Honoré, and Guillaume Pallez. Profiles of upcoming hpc applications and their impact on reservation strategies. *IEEE Transactions on Parallel and Distributed Systems*, 32(5):1178–1190, 2020.

[J6] Julien Herrmann and Guillaume Pallez. H-revolve: A framework for adjoint computation on synchronous hierarchical platforms. *ACM Transactions on Mathematical Software (TOMS)*, 46(2):1–25, 2020.

[J7] Guillaume Aupy, Anne Benoit, Brice Goglin, Loïc Pottier, and Yves Robert. Co-scheduling hpc workloads on cache-partitioned cmp platforms. *The International Journal of High Performance Computing Applications*, 33(6):1221–1239, 2019.

[J8] Guillaume Aupy, Brice Goglin, Valentin Honoré, and Bruno Raffin. Modeling high-throughput applications for in situ analytics. *The International Journal of High Performance Computing Applications*, 33(6):1185–1200, 2019.

[J9] Ana Gainaru, Valentin Le Fèvre, and Guillaume Pallez. I/o scheduling strategy for periodic applications. *ACM Transactions on Parallel Computing (TOPC)*, 6(2):1–26, 2019.

[J10] Ana Gainaru, Hongyang Sun, Guillaume Aupy, Yuankai Huo, Bennett A Landman, and Padma Raghavan. On-the-fly scheduling versus reservation-based scheduling for unpredictable workflows. *The International Journal of High Performance Computing Applications*, 33(6):1140–1158, 2019.

[J11] Tien-Dat Phan, Guillaume Pallez, Shadi Ibrahim, and Padma Raghavan. A new framework for evaluating straggler detection mechanisms in mapreduce. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 2019.

[J12] Guillaume Aupy, Anne Benoit, Sicheng Dai, Loïc Pottier, Padma Raghavan, Yves Robert, and Manu Shantharam. Co-scheduling amdahl applications on cache-partitioned systems. *IJHPCA*, 32(1):123–138, 2018.

[J13] Guillaume Aupy and Julien Herrmann. Periodicity in optimal hierarchical checkpointing schemes for adjoint computations. *Optimization Methods and Software*, 32(3):594–624, 2017.

[J14] Guillaume Aupy and Anne Benoit. Approximation algorithms for energy, reliability, and makespan optimization problems. *Parallel Processing Letters*, 26(1), 2016.

[J15] Guillaume Aupy, Anne Benoit, Henri Casanova, and Yves Robert. Checkpointing strategies for scheduling computational workflows. *International Journal of Networking and Computing*, 6(1):2–26, 2016. hal-01251939. Journal version of [C22].

[J16] Guillaume Aupy, Julien Herrmann, Paul Hovland, and Yves Robert. Optimal multistage algorithm for adjoint computation. *SIAM J. Scientific Computing*, 38(3), 2016.

[J17] Guillaume Aupy, Manu Shantharam, Anne Benoit, Yves Robert, and Padma Raghavan. Co-scheduling algorithms for high-throughput workload execution. *J. Scheduling*, 19(6):627–640, 2016.

[J18] Guillaume Aupy, Anne Benoit, Matthieu Journault, and Yves Robert. Power-aware replica placement in tree networks with multiple servers per client. *Sustainable Computing: Informatics and Systems*, 5:41–53, 2015. hal-01059365. Journal version of [C25].

[J19] Guillaume Aupy, Yves Robert, Frédéric Vivien, and Dounia Zaidouni. Checkpointing algorithms and fault prediction. *J. Parallel Distrib. Comput.*, 74(2):2048–2064, 2014. hal-00788313v2.

[J20] Guillaume Aupy, Anne Benoit, Fanny Dufossé, and Yves Robert. Reclaiming the energy of a schedule: models and algorithms. *Concurrency and Computation: Practice and Experience*, 25(11):1505–1523, 2013. hal-00763388.

[J21] Guillaume Aupy and Olivier Bournez. On the number of binary-minded individuals required to compute sqrt(1/2). *Theor. Comput. Sci.*, 412(22):2262–2267, 2011. hal-00760928.

## Articles in international refereed conferences

*I used to separate workshop proceedings from conferences proceedings, but recently I found that I was increasingly disappointed by the reviewing process in major conferences, and on the contrary enjoyed smaller workshops with more heart to heart discussions (such as JSSPP), hence they are not separated anymore.*

[C1] Robin Boëzennec, Fanny Dufossé, and Guillaume Pallez. Optimization metrics for the evaluation of batch schedulers in hpc. In *26th edition of the workshop on Job Scheduling Strategies for Parallel Processing*, 2023.

[C2] Francieli Boito, Guillaume Pallez, and Luan Teylo. The role of storage target allocation in applications' i/o performance with beegfs. In *CLUSTER 2022-IEEE International Conference on Cluster Computing*, 2022.

[C3] Yiqin Gao, Guillaume Pallez, Yves Robert, and Frédéric Vivien. Work-in-progress: Evaluating task dropping strategies for overloaded real-time systems. In *42nd IEEE Real-Time Systems Symposium, RTSS 2021, Dortmund, Germany, December 7-10, 2021*, pages 528–531. IEEE, 2021.

[C4] Jesus Carretero, Emmanuel Jeannot, Guillaume Pallez, David E Singh, and Nicolas Vidal. Mapping and scheduling hpc applications for optimizing i/o. In *Proceedings of the 34th ACM International Conference on Supercomputing*, pages 1–12, 2020.

[C5] Yishu Du, Loris Marchal, Guillaume Pallez, and Yves Robert. Robustness of the young/daly formula for stochastic iterative applications. In *49th International Conference on Parallel Processing-ICPP*, pages 1–11, 2020.

[C6] Ana Gainaru, Brice Goglin, Valentin Honoré, Guillaume Pallez Aupy, Padma Raghavan, Yves Robert, and Hongyang Sun. Reservation and checkpointing strategies for stochastic jobs. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 853–863. IEEE, 2020.

[C7] Massinissa Ait Aba, Guillaume Aupy, and Alix Munier-Kordon. Scheduling on two unbounded resources with communication costs. In *Euro-Par 2019: Parallel Processing - 25th International Conference on Parallel and Distributed Computing, Proceedings*, 2019.

[C8] Guillaume Aupy, Olivier Beaumont, and Lionel Eyraud-Dubois. Sizing and partitioning strategies for burst-buffers to reduce IO contention. In *2019 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2019*, 2019.

[C9] Guillaume Aupy, Ana Gainaru, Valentin Honoré, Padma Raghavan, Yves Robert, and Hongyang Sun. Reservation strategies for stochastic jobs. In *2019 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2019*, 2019.

[C10] Ana Gainaru and Guillaume Pallez. Making speculative scheduling robust to incomplete data. In *2019 IEEE/ACM 10th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*, pages 62–71. IEEE, 2019.

[C11] Ana Gainaru, Guillaume Pallez, Hongyang Sun, and Padma Raghavan. Speculative scheduling for stochastic hpc applications. In *48th International Conference on Parallel Processing, ICPP 2019, Kyoto, Japan, 2019*, 2019.

[C12] Guillaume Aupy, Olivier Beaumont, and Lionel Eyraud-Dubois. What size should your buffers to disk be? In *2018 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2018*, 2018.

[C13] Guillaume Aupy, Anne Benoit, Brice Goglin, Loïc Pottier, and Yves Robert. Co-scheduling hpc workloads on cache-partitioned cmp platforms. In *2018 IEEE International Conference on Cluster Computing, CLUSTER*, 2018.

[C14] Hongyang Sun, Redouane Elghazi, Ana Gainaru, Guillaume Aupy, and Padma Raghavan. Scheduling parallel tasks under multiple resources: List scheduling vs. pack scheduling. In *2018 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2018*, 2018.

[C15] Guillaume Aupy, Anne Benoit, Loïc Pottier, Padma Raghavan, Yves Robert, and Manu Shantharam. Co-scheduling algorithms for cache-partitioned systems. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPS Workshops 2017, Orlando / Buena Vista, FL, USA, May 29 - June 2, 2017*, pages 874–883, 2017.

[C16] Guillaume Aupy, Clement Brasseur, and Loris Marchal. Dynamic memory-aware task-tree scheduling. In *2017 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2017, Orlando, FL, USA, May 29 - June 2, 2017*, pages 758–767, 2017.

[C17] Guillaume Aupy, Ana Gainaru, and Valentin Le Févre. Periodic i/o scheduling for supercomputers. In *High Performance Computing Systems. Performance Modeling, Benchmarking and Simulation - 8th International Workshop, PMBS 2017, Denver, CO, USA, November 13, 2017.*, 2017.

[C18] Guillaume Aupy, Yves Robert, and Frédéric Vivien. Assuming failure independence: Are we right to be wrong? In *2017 IEEE International Conference on Cluster Computing, CLUSTER 2017, Honolulu, HI, USA, September 5-8, 2017*, pages 709–716, 2017.

[C19] Tien-Dat Phan, Shadi Ibrahim, Amelie Chi Zhou, Guillaume Aupy, and Gabriel Antoniu. Energy-driven straggler mitigation in mapreduce. In *Euro-Par 2017: Parallel Processing - 23rd International Conference on Parallel and Distributed Computing, Santiago de Compostela, Spain, August 28 - September 1, 2017, Proceedings*, pages 385–398, 2017.

[C20] Shashank Shekhar, Ajay Dev Chhokra, Anirban Bhattacharjee, Guillaume Aupy, and Aniruddha S. Gokhale. INDICES: exploiting edge resources for performance-aware cloud-hosted services. In *1st IEEE International Conference on Fog and Edge Computing, ICFEC 2017, Madrid, Spain, May 14-15, 2017*, pages 75–80, 2017.

[C21] Guillaume Aupy, JeongHyung Park, and Padma Raghavan. Locality-aware laplacian mesh smoothing. In *45th International Conference on Parallel Processing, ICPP 2016, Philadelphia PA, USA, August 16-19, 2016*, pages 588–597. IEEE, 2016.

[C22] Guillaume Aupy, Anne Benoit, Henri Casanova, and Yves Robert. Scheduling computational workflows on failure-prone platforms. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop, IPDPS 2015, Hyderabad, India, May 25-29, 2015*, pages 473–482, 2015. hal-01075100.

[C23] Ana Gainaru*, Guillaume Aupy*, Anne Benoit, Franck Cappello, Yves Robert, and Marc Snir. Scheduling the I/O of HPC applications under congestion. In *2015 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2015, Hyderabad, India, May 25-29, 2015*, pages 1013–1022, 2015. *These authors contributed equally. hal-00983789.

[C24] Humayun Kabir, Joshua Dennis Booth, Guillaume Aupy, Anne Benoit, Yves Robert, and Padma Raghavan. Sts-k: a multilevel sparse triangular solution scheme for NUMA multicores. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2015, Austin, TX, USA, November 15-20, 2015*, pages 55:1–55:11, 2015. hal-01183904.

[C25] Guillaume Aupy, Anne Benoit, Matthieu Journault, and Yves Robert. Power-aware replica placement in tree networks with multiple servers per client. In *Euro-Par 2014 Parallel Processing - 20th International Conference, Porto, Portugal, August 25-29, 2014. Proceedings*, pages 608–619, 2014. hal-01059365.

[C26] Guillaume Aupy, Anne Benoit, Thomas Hérault, Yves Robert, and Jack J. Dongarra. Optimal checkpointing period: Time vs. energy. In *High Performance Computing Systems. Performance Modeling, Benchmarking and Simulation - 4th International Workshop, PMBS 2013, Denver, CO, USA, November 18, 2013. Revised Selected Papers*, pages 203–214, 2013. hal-00926199.

[C27] Guillaume Aupy, Anne Benoit, Thomas Hérault, Yves Robert, Frédéric Vivien, and Dounia Zaidouni. On the combination of silent error detection and checkpointing. In *IEEE 19th Pacific Rim International Symposium on Dependable Computing, PRDC 2013, Vancouver, BC, Canada, December 2-4, 2013*, pages 11–20, 2013. hal-00836871.

[C28] Guillaume Aupy, Anne Benoit, Rami G. Melhem, Paul Renaud-Goud, and Yves Robert. Energy-aware checkpointing of divisible tasks with soft or hard deadlines. In *International Green Computing Conference, IGCC 2013, Arlington, VA, USA, June 27-29, 2013, Proceedings*, pages 1–8, 2013. hal-00857244.

[C29] Guillaume Aupy, Mathieu Faverge, Yves Robert, Jakub Kurzak, Piotr Luszczek, and Jack Dongarra. Implementing a systolic algorithm for QR factorization on multicore clusters with parsec. In *Euro-Par 2013: Parallel Processing Workshops - PROPER 2013, Aachen, Germany, August 26-27, 2013. Revised Selected Papers*, pages 657–667, 2013. hal-00879248.

[C30] Guillaume Aupy, Yves Robert, Frédéric Vivien, and Dounia Zaidouni. Checkpointing strategies with prediction windows. In *IEEE 19th Pacific Rim International Symposium on Dependable Computing, PRDC 2013, Vancouver, BC, Canada, December 2-4, 2013*, pages 1–10, 2013. hal-00847622.

[C31] Guillaume Aupy, Anne Benoit, and Yves Robert. Energy-aware scheduling under reliability and makespan constraints. In *19th International Conference on High Performance Computing, HiPC 2012, Pune, India, December 18-22, 2012*, pages 1–10, 2012. hal-00763384.

[C32] Leon Atkins, Guillaume Aupy, Daniel Cole, and Kirk Pruhs. Speed scaling to manage temperature. In *Theory and Practice of Algorithms in (Computer) Systems (TAPAS)*, pages 9–20, 2011. hal-00786200.

[C33] Guillaume Aupy, Anne Benoit, Fanny Dufossé, and Yves Robert. Brief announcement: Reclaiming the energy of a schedule, models and algorithms. In *23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), San Jose, CA, USA, June 4-6, 2011*, pages 135–136, 2011.

## Other type of publications

[O1] Frédéric Blanqui, Anne Canteaut, Hidde de Jong, Sébastien Imperiale, Nathalie Mitton, Guillaume Pallez, Xavier Pennec, Xavier Rival, and Bertrand Thirion. Recommandations sur les «éditeurs de la zone grise». *Rapports de la Commission d'Évaluation*, 2023.

[O2] Luce Brotcorne, Anne Canteaut, Aline Viana, Céline Grandmont, Benjamin Guedj, Stéphane Huot, Valérie Issarny, Guillaume Pallez, Valérie Perrier, Vivien Quema, et al. Indicateurs de suivi de l'activité scientifique de l'inria. *Rapports de la Commission d'Évaluation*, 2020.

[O3] Guillaume Aupy and Yves Robert. Scheduling for fault-tolerance: An introduction. In Sushil K. Prasad, Anshul Gupta, Arnold L. Rosenberg, Alan Sussman, and Charles C. Weems, editors,

*Topics in Parallel and Distributed Computing, Enhancing the Undergraduate Curriculum: Performance, Concurrency, and Programming on Modern Platforms*, pages 143–170. Springer, 2018.

## Research reports

[RR1] Ahmad Tarraf, Alexis Bandet, Francieli Boito, Guillaume Pallez, and Felix Wolf. FTIO: Detecting I/O Periodicity Using Frequency Techniques. Preprint available at https://arxiv.org/abs/2306.08601, 2023.

[RR2] Yishu Du, Loris Marchal, Guillaume Pallez, and Yves Robert. Doing better for jobs that failed: node stealing from a batch scheduler's perspective. Preprint available at https://inria.hal.science/hal-03643403, 2022.

[RR3] Francieli Zanon Boito, Guillaume Pallez, Luan Teylo, and Nicolas Vidal. IO-SETS: Simple and efficient approaches for i/o bandwidth management. Preprint available at https://inria.hal.science/hal-03648225/, 2022.

# CV


## GUILLAUME PALLEZ
**Researcher (CRCN)**

Inria Bordeaux Sud-Ouest
200 avenue de la vieille tour.
33405 Talence Cedex, France
Phone: +33 5 24 57 40 64

Mail: guillaume.pallez@inria.fr
Web: http://people.bordeaux.inria.fr/gaupy/


## Positions

| | | |
|---|---|---|
| **Dec. 2016–** | **Researcher (CRCN),** | **INRIA Bordeaux (France)** |
| *2015–2016* | Research Assistant Professor | Penn. State Univ / Vanderbilt University (USA) |
| 2014 | Visiting Scolar | Argonne National Laboratory (USA) |

## Education

| | |
|---|---|
| 2011 - 2014 | **PhD in Computer Science**, ENS Lyon<br>Title: *Resilient and energy-efficient scheduling algorithms at scale*<br>Defended on Sept. 16, 2014 @ENS Lyon; [A. Benoit, Y. Robert (Advisors)]<br>Available http://people.bordeaux.inria.fr/gaupy/thesis.html |
| 2010 - 2011 | **M2 Theoretical Computer Science**, Parisian Master of Research in Computer Science (MPRI), Paris, France |
| 2009 - 2010 | **M1 Theoretical Computer Science**, ENS Lyon, France |

## Award

2019 IEEE CS TCHPC Early Career Researchers Award for Excellence in HPC

## Selected publications

Selection of recent publications I like, in reverse chronological order. In general the author order is alphabetical. In 2019 I changed my last name from Aupy to Pallez.

1. **Optimization Metrics for the Evaluation of Batch Schedulers in HPC.** R. Boezennec, F. Dufossé, G. Pallez. In 26th edition of the workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), 2023.

2. **Profiles of Upcoming HPC Applications and Their Impact on Reservation Strategies.** A. Gainaru, B. Goglin, V. Honoré, G. Pallez. In IEEE Transactions on Parallel and Distributed Systems (TPDS), 2021.

3. **H-Revolve: A Framework for Adjoint Computation on Synchrone Hierarchical Platforms.** J. Herrmann, G. Pallez. In Transactions of Mathematical Software (TOMS), 2020.

4. **Reservation and Checkpointing Strategies for Stochastic Jobs.** A. Gainaru, B. Goglin V. Honoré, G. Pallez P. Raghavan, Y. Robert, HY. Sun. In IEEE International Parallel and Distributed Processing Symposium, IPDPS 2020.

5. **What size should your buffers to disk be?** G. Aupy, O. Beaumont, and L. Eyraud-Dubois. In IEEE International Parallel and Distributed Processing Symposium, IPDPS 2018.

## Funded Proposal

NewPAs: New Portrayal of HPC Applications (PI). Inria Exploratory Action, 2022 - 2025.

Admire: Adaptative multi-tier intelligent data manager for Exascale (Participant). Euro-HPC, 2021 - 2025.

ADML: Automatic Differentiation for Machine Learning (co-PI). FACCTS (collaboration with Paul Hovland, ANL), 2018-2020.

Dash: Data-Aware Scheduling at Higher scale (PI). ANR JCJC, 2018 - 2023.

## Supervision

| Period | Name | Status | Co-advisor | Now? |
|---|---|---|---|---|
| 2022 - *Now* | R. Boezennec, | **PhD student** | F. Dufossé | |
| 2021 - *Now* | A. Bandet, | **PhD student** | F. Zanon-Boito | |
| 2021 - 2023 | L. Teylo, | **Postdoctoral fellow** | F. Zanon-Boito | Inria researcher |
| 2018 - 2019 | J. Herrmann, | **Postdoctoral fellow** | | CNRS researcher |
| 2018 - 2022 | N. Vidal, | **PhD student** | E. Jeannot | Postdoc ORNL (US) |
| 2017 - 2020 | V. Honoré, | **PhD student** | B. Goglin | Associate Professor, Evry |

I have also supervised Master and undergrad students. I have served on two PhD examinations: Sorbonne Univ. (FR); Imperial College of London (UK).

## Selected Teachings and outreach

- During the Falls 2019, 2020 and 2021, I have co-teached a graduate class (Master 2, about 25 students): *Algorithms for High-Performance Computing Platforms* (Enseirb-Matmeca).

- *A Supercomputer in my kitchen* (2016-2022): popularization talk given frequently to middle school, high school and undergrad students for the past years.

- *Le non sens écologique des voitures autonomes* (2019), Blog post (Binaire, hosted by *Le Monde*) about autonomous car versus ecology.

- Logique juridique et logique mathématique : quelles convergences ?, Aupy and Platon, Actes du colloque des Convergences du Droit et du Numérique, 2017: Joint work with a Public Law professor to present how one could model mathematically legal logic/reasonning (part of the meeting *Convergences du droit et du numérique*).

# Collective responsibilities

## Selected Institutional responsibilities

2019-2023 **Elected member of the Research Evaluation Committee of Inria**
This commitee is in charge of all types of scientific evaluation at Inria (including prospective work, recruitment, team evaluation, advancement etc.). Example of specific actions:
Co-author of the report *Indicators for monitoring Inria's scientific activity*, targetted for the French Ministry of Research (2020). https://hal.inria.fr/hal-03033764/
Lead author of the report *Recommandations on "grey-zone editors"* (2022). https://inria.hal.science/hal-04001505/

2019-2022 **Staff representative at CHSCT**
The CHSCT is a legal commission dedicated to setting a "healthy and safe work environment". Example of specific actions:
1. Proposing and evaluating measures to deal with the Covid situation at Inria Bordeaux;
2. Co-writing the *Inria global Action plan on F/M professional equality* (2020).

## Selected Organization of Scientific Meetings

I have had many major responsibilities, particularly within the Supercomputing conference (SC): the main event for the HPC communities, attended by 13k people (academics, users, system administrators, industry).

| | |
|---|---|
| 2024 | **IEEE/ACM Supercomputing (SC'24)**; US, expected $13k$ participants<br>Technical Program chair (VC: Judith Hill), supervising the whole technical program of SC (Papers, Posters, Tutorials, Workshops, Panels, Awards etc). |
| 2023 | **IEEE Cluster Conference (Cluster'23)**; US, exp. $\sim$ 200 part.<br>Finance chair – Executive committee, budget of approx USD 200k. |
| 2022 | **International Conference on Parallel Processing (ICPP'22)**; Virtual, exp. $\sim$ 200 part.<br>Co-general chair – with E. Jeannot |
| 2020 | **IEEE/ACM Supercomputing (SC'20)**; US, $\sim 13k$ part.<br>Finance liaison to Tech Program |
| 2018 | **IEEE/ACM Supercomputing (SC'18)**; US, $\sim 12k$ part.<br>Inclusivity vice-chair – with T. Collis, supervising and organizing all activities for accessibility and diversity at SC) |
| 2017 | **IEEE/ACM Supercomputing (SC'17)**; US, $\sim 12k$ part.<br>Technical Program vice-chair – with P. Raghavan. |

## Selected Evaluations of Science

2022 **IEEE Cluster Conference (Cluster'22)**; DE, exp. $\sim$ 200 participants
Panel co-chair – Panels are a new program for Cluster conference, introduced in 2022.

2021 **International Conference on Parallel Processing (ICPP'21)**; US, exp. $\sim$ 200 part.
Algorithm track co-chair – with M. Kneppley

2020-2022 **IEEE Transactions on Parallel and Distributed Systems (TPDS)**
Review Board Member – set of trusted experts in the area of focus of the journal TPDS

2018 **IEEE/ACM Supercomputing (SC'18)**; US, $\sim 13k$ part. ($\sim$6.5k for Tech Program)
Workshop chair – in charge of workshop selection process and organization; $\sim$ 35 workshops

2018 **International Conference on Parallel Processing (ICPP'18)**; US, $\sim$ 200 part.
Algorithm track vice chair – with K. Madduri

2017 **International Conf. on Algorithms and Architectures for Parallel Processing (ICA3PP'17)**; Sweden, $\sim$ 100 part.
Parallel and Distributed Algorithms track co-chair – with X. Tang

- *Program Committee member (selected)*: Siam ACDA (2021), ICPP (2017, 2019, 2020), IEEE IPDPS (2016, 2020, 2021, 2022, 2023), IEEE/ACM SC [Papers (2016, 2017), Best Paper (2020), Tutorials (2019, 2020)].
- *Scientific Proposal Reviewer*: French National Science Foundation (ANR, external reviewer)
- *Editorship*: Concurrency and Computation: Practice and Experience (CCPE, Guest Editor)
- Member of the *Inria Research Evaluation Committee*

## Other International Service

2020-2022 **IEEE TCPP Executive Committee**
Member of the Executive Commitee of IEEE Technical Committee on Parallel Processing (TCPP). In charge of all web communication and coordination between the different conferences organized by TCPP (IPDPS, PerCom, PACT, HiPC, etc).

# Résumé en français

Les machines de Calcul à Haute-Performances (HPC) sont des ressources de calculs extrèmement intenses. Pour donner un ordre de grandeur, la machine[10] la plus puissante actuellement (Juin 2023) est *Frontier* à Oak Ridge National Lab. Cette machine contient environ 8.7 millions de cœurs pour une performance théorique de 1.7EFlop/s. Elle a une consommation de 21MW.

Le sujet de mes recherches de ces 10 dernières années (et depuis l'obtention de ma thèse de doctorat) a porté très largement sur la problématique de la gestion des ressources en HPC. Spécifiquement :

**L'ordonnancement d'I/O** [C2, J9, J3, RR3, C4, C8, C12]: Une problématique majeure des super ordinateurs est la gestion des données générées qui doivent être stockées sur les disques (par exemple données de visualisation, checkpoints pour la résilience). Une part de ma recherche s'est concentrée sur des techniques d'ordonnancement pour minimiser la contention I/O et améliorer la performance d'applications HPC.

**Ordonnancement d'applications à performances stochastiques** [J5, C6, C9, C11, C10, J10]: Avec la généralisation du HPC, de nouveau type d'applications sont exécutés sur les super-ordinateurs. Ces applications ont des caractéristiques différentes des applications typiques HPC. Particulièrement leur temps d'exécution est moins déterministe. Dans ce projet mené avec des scientifiques de l'université de Nashville, nous avons étudié certaines applications de neurosciences et avons modélisé leur performance en utilisant des outils probabilistes. Nous nous intéressons aux performances des gestionnaires de ressources face à ces nouveaux profils et avons proposé de nouvelles stratégies algorithmiques pour gérer ces applications "stochastiques".

**Calcul d'adjoint sous contraintes mémoires** [J6, J4, J13, J16]: Pour ce projet mené avec des collègues d'Argonne National Lab. (Paul Hovland, Krishna Narayanan), nous avons étudié un type de graphe spécifique : les graphes de rétropropagation (ou différentiation automatique). Nous avons proposé de nouvelles solutions algorithmiques pour les exécuter en présence de contraintes mémoires. Avec Julien Herrmann, nous avons ensuite développé une librairie Python (H-Revolve) où ces algorithmes ont été implémentés.

**Résilience** [C18, J2, C5, RR2]: Depuis ma thèse je me suis intéressé à la gestion des fautes sur les ressources HPC. Le checkpoint est la technique la plus habituelle pour protéger les applications HPC: après chaque faute, l'application qui s'exécutait sur la ressource défaillante est interrompue et doit être redémarrée. Sans checkpoint le travail exécuté serait perdu. En présence de checkpoint, l'application peut redémarrer à son dernier point de sauvegarde après un temps de réjunévation. Récemment je me suis particulièrement intéressé aux limitations des modèles de checkpoints.

Le point commun entre ces études est l'approche méthodologique que j'apporte en les étudiant. Ma recherche se concentre particulièrement sur la modélisation des problèmes et le fait d'y proposer des

[10] dont les données sont publiques

solutions algorithmiques. En revenant sur 10 ans de recherche, j'ai pu observé l'évolution et la maturité que j'ai obtenu sur la problématique de modéliser les performances d'un gestionnaire de ressources. C'est ce que je vais essayer de décrire dans ce document.

**Contenu de ce document:** Je fais l'hypothèse que l'utilisation de ressources des différents jobs ainsi que leur variations temporelles possèdent une part d'aléa. La variabilité de ces besoins leur est liée et peut être très large. À partir de là, je pense que les algorithmes d'ordonnancement et les logiciels pour l'HPC doivent embrasser cet aléa. Actuellement, il me semble que cette incertitude n'est pas encore prise en compte de manière satisfaisante dans la conception de ces solutions.

Dans ce document, je défends de nouvelles manières d'étudier et d'incorporer l'imprécision des performance dans les modèles utilisés pour la conception de gestionnaires de ressources. Cette démonstration se basera sur la recherche que j'ai effectuée ces 10 dernières années.

Au travers de questions et d'exemples spécifiques, j'essaye de proposer dans ce document comment on pourrait (i) concevoir un modèle qui représente de manière plus fidèle les applications tout en restant pratiques ; puis (ii) je montre l'importance de questionner les limites des modèles proposé en en testant certains contre des hypothèses du "monde réel".

Ce document est ensuite conçu ainsi:

- Dans le Chapitre 1, je présente des remarques introductives sur la conception de modèle pour l'algorithmique de gestionnaires de ressources. En particulier je discute des hypothèse courantes faites dans la conception d'un modèle.
- Dans le Chapitre 2, au travers d'un cas pratique (la tolérance aux fautes), je montre comment on pourrait détecter des imprécisions d'un modèle.
- Cela mène dans le Chapitre 3 à réouvrir les questions de fond sur la conception de modèles. Par ailleurs, je plaide dans ce chapitre pour une compréhension plus qualitative des objectifs d'évaluation plutôt que de considérer des objectifs quantitatifs de minimisation/maximisation.
- Dans le Chapitre 4, au travers d'un cas pratique (l'ordonnancement d'I/O), je discute en profondeur une recherche pour des modèles pratiques. Spécifiquement, je montre que les modèles peuvent différer en fonction des besoins de ce qu'on cherche à étudier.
- Dans le Chapitre 5, je discute à nouveau ces questions sur un autre cas pratique: l'incertitude pour les temps d'éxecutions de certains jobs.
- Enfin, après quelques remarques de conclusions, je discute les grandes lignes de ce qu'il me semble être d'importantes directions de recherche dans le futur.