Communauté
UNIVERSITÉ Grenoble Alpes

**THÈSE**

Pour obtenir le grade de

**DOCTEUR DE LA**
**COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES**

Spécialité : **Informatique**

Arrêté ministériel : 25 mai 2016

Présentée par

**Michael Mercier**

Thèse dirigée par **Bruno Raffin**
et codirigée par **Olivier Richard**

préparée au sein du **Laboratoire d'Informatique de Grenoble**
dans **l'École Doctorale Mathématiques, Sciences et technologies de l'information, Informatique**

# Contribution à la convergence d'infrastructure entre le calcul haute performance et le traitement de données a large échelle

# Contribution to High Performance Computing and Big Data Infrastructure Convergence

Thèse soutenue publiquement le **1er juillet 2019**,
devant le jury composé de :

# Abstract

The amount of produced data, either in the scientific community or the commercial world, is constantly growing. The field of Big Data has emerged to handle large amounts of data on distributed computing infrastructures. High-Performance Computing (HPC) infrastructures are traditionally used for the execution of compute intensive workloads. However, the HPC community is also facing an increasing need to process large amounts of data derived from high definition sensors and large physics apparati. The convergence of the two fields -HPC and Big Data- is currently taking place. In fact, the HPC community already uses Big Data tools, which are not always integrated correctly, especially at the level of the file system and the Resource and Job Management System (RJMS).

In order to understand how we can leverage HPC clusters for Big Data usage, and what are the challenges for the HPC infrastructures, we have studied multiple aspects of the convergence: We initially provide a survey on the software provisioning methods, with a focus on data-intensive applications. We contribute a new RJMS collaboration technique called BeBiDa which is based on 50 lines of code whereas similar solutions use at least 1000 times more. We evaluate this mechanism on real conditions and in simulated environment with our simulator Batsim. Furthermore, we provide extensions to Batsim to support I/O, and showcase the developments of a generic file system model along with a Big Data application model. This allows us to complement BeBiDa real conditions experiments with simulations while enabling us to study file system dimensioning and trade-offs.

All the experiments and analysis of this work have been done with reproducibility in mind. Based on this experience, we propose to integrate the development workflow and data analysis in the reproducibility mindset, and give feedback on our experiences with a list of best practices.

# Résumé

La quantité de données produites, que ce soit dans la communauté scientifique ou commerciale, est en croissance constante. Le domaine du Big Data a émergé face au traitement de grandes quantités de données sur les infrastructures informatiques distribuées. Les infrastructures de calcul haute performance (HPC) sont traditionnellement utilisées pour l'exécution de charges de travail intensives en calcul. Cependant, la communauté HPC fait également face à un nombre croissant de besoin de traitement de grandes quantités de données dérivées de capteurs haute définition et de grands appareils physique. La convergence des deux domaines -HPC et Big Data- est en cours. En fait, la communauté HPC utilise déjà des outils Big Data, qui ne sont pas toujours correctement intégrés, en particulier au niveau du système de fichiers ainsi que du système de gestion des ressources (RJMS).

Afin de comprendre comment nous pouvons tirer parti des clusters HPC pour l'utilisation du Big Data, et quels sont les défis pour les infrastructures HPC, nous avons étudié plusieurs aspects de la convergence: nous avons d'abord proposé une étude sur les méthodes de provisionnement logiciel, en mettant l'accent sur les applications utilisant beaucoup de données. Nous contribuons a l'état de l'art avec une nouvelle technique de collaboration entre RJMS appelée BeBiDa basée sur 50 lignes de code alors que des solutions similaires en utilisent au moins 1000 fois plus. Nous évaluons ce mécanisme en conditions réelles et en environnement simulé avec notre simulateur Batsim. En outre, nous fournissons des extensions à Batsim pour prendre en charge les entrées/sorties et présentons le développements d'un modèle de système de fichiers générique accompagné d'un modèle d'application Big Data. Cela nous permet de compléter les expériences en conditions réelles de BeBiDa en simulation tout en étudiant le dimensionnement et les différents compromis autours des systèmes de fichiers.

Toutes les expériences et analyses de ce travail ont été effectuées avec la reproductibilité à l'esprit. Sur la base de cette expérience, nous proposons d'intégrer le flux de travail du développement et de l'analyse des données dans l'esprit de la reproductibilité, et de donner un retour sur nos expériences avec une liste de bonnes pratiques.

# Contents

# 1 Introduction

The rise of the Internet provoked an explosion of data produced in the world since the beginning of 2000. Figure 1.1 illustrates this continuously accelerating growth; The global internet traffic was around 20 EB per month in 2010 and it is for 2017 over 120 EB per month (6x in 7 years).

The traditional way to store data on a database, hosted by one big machine, is not suitable anymore. The volume and variety of data, along with the velocity of its production, requires to distribute the process across a whole infrastructure. The field of Big Data was born in large Web companies to tackle this problem. Data has to be collected, stored, indexed, and analyzed with a set of interconnected tools, in a distributed way and at large scale.

On the other hand, we also see the exponential growth, depicted in Figure 1.2, of the computation power of the biggest computers in the world: the Supercomputers.

We are now on the road to the next change of scale with the future High-Performance Computing (HPC) machines: the exascale, where one supercomputer is capable to compute at the exaFLOPS scale ($10^{18}$ floating operations per second). The past few years, these two separate fields are in a process of convergence. In the Big Data community, there is recently a lot of traction towards High-performance computers mainly because of the popularity of machine learning, and in particular the deep learning, which is computation intensive. In the HPC community, which is traditionally computation centric, we are currently witnessing the rise of the amount of data-intensive workloads that have to be managed by the HPC platform since this is becoming the saturation point for the conventional infrastructures. In fact, the scientific communities, which are currently the principal HPC users, are facing the emergence of large apparati which produce an unprecedented amount of data: For example, the Large Hadron Collider (LHC) is the world's largest particle collider that has generated around 50 PetaBytes of data in 2018 [CER18]. In general, most of the scientific fields are now starting to leverage the enormous amounts of data which are produced and captured. HPC platforms are already complex, with multiple levels of hardware, software and services: from the applications that run into it, to the resource and job management systems that orchestrate

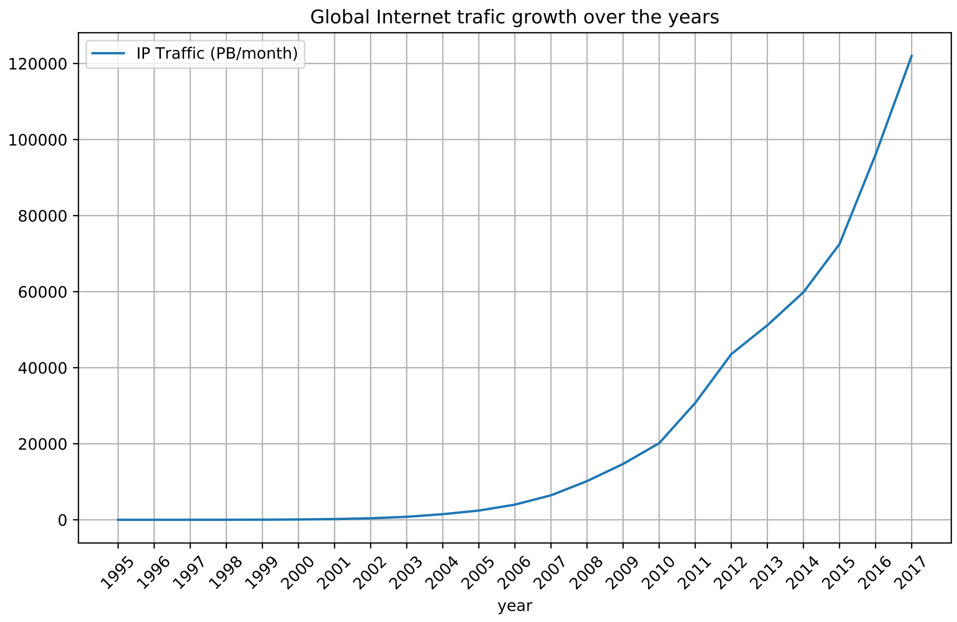


**Figure 1.1.:** Global Internet Traffic over the years [Wik19c].

the applications and manage the resources sharing. We are now facing a new complexity related to the new data-oriented usage of those platforms. In this thesis we are focusing on the infrastructure convergence between Big Data and HPC platforms: What are the requirements of hybrid workloads, composed of computation-centric and data-centric applications? What kind of platform is the most suited for this kind of workload in terms of infrastructure architecture and services?

To be able to answer these questions we need to state clear definitions of both fields regarding the workload and the computer system infrastructure. Chapter 2 describes HPC and Big Data infrastructures from the applications to the resource management system. It also explores the convergence points between them at each level of these infrastructures.

Big Data and HPC infrastructures are very different. Big Data infrastructures are generally composed of relatively cheap commodity hardware that can scale upto 10,000 machines per cluster. Since the hardware is unreliable, the software needs to be resilient to machine failures. Big Data resilience is implemented at all software levels of the Big Data infrastructure and requires a lot of interconnected services.

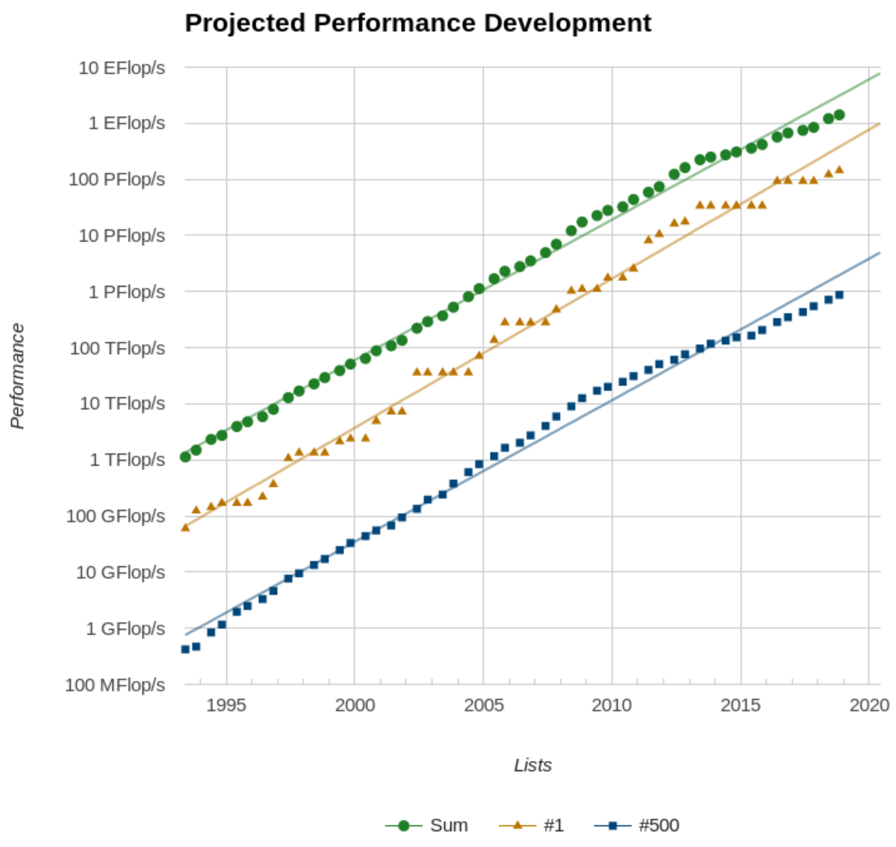


**Figure 1.2.:** Computation capacity growth over the years [top19].

In contrast, the HPC infrastructure is composed of state-of-the-art hardware with redundancy to minimize risks for 'down-time' as much as possible. Most of the software is not resilient to avoid the cost of managing dynamicity. Furthermore, the number of services that run on an HPC platform is kept to a minimum to avoid applications' perturbation.

One way to merge the two infrastructures is to use Big Data applications on HPC hardware. However, this leads to a new problem: How can a complex, service-based, Big Data software stack be used in an HPC platform made for batch processing? Chapter 3 provides a survey on the various existing approaches. It contains a definition of the users needs and compares the different applications' deployment strategies, called the software stack provisioning, which fulfill these needs.

On both HPC and Big Data systems, computer clusters are managed by a resource and job management system (RJMS). In Chapter 4, we propose a simple, yet powerful, approach called BeBiDa which enables Big Data and HPC RJMS collaboration. We evaluate this approach with real-conditions' experiments. This is the first principal contribution of this thesis.

Experimentation on real machines with the joined complexity of both software stacks is complex and costs a lot of human and electrical energy. Chapter 5 exposes the design of our I/O aware infrastructure simulator called Batsim. This is the second principal contribution of this thesis. This simulator enables to experiment on hybrid HPC and Big Data systems in simulated environment. It allows us to reproduce and extend the experiments carried out in real-conditions. The results of these simulations corroborate our analysis related to the previous findings and thus partly validate the simulator.

Finally, Chapter 6 describe the scientific methodologies that were used and explored during this thesis along with the tools that were developed and used to elaborate the results that are presented in this manuscript. It includes a reflection on reproducible software environments while highlighting its importance in the context of experimentation under realistic conditions.

# 2 HPC and Big Data: Definition, Comparison, and Convergence

## Chapter's Abstract

The most preeminent difference between HPC and Big Data is their purposes: Big Data is a field of research that was invented to distribute the processing of large amounts of data, while HPC is simply the more advanced computer for parallel computation. But, in order to clarify what really is Big Data and what is HPC, we need to define them precisely. First, what those terms refer to? They both defined a field of research in Computer Science, closely related to industrial and production concerns. But, their purpose is different. This chapter is an attempt to define those terms, beyond the buzz words, by comparing them and expose their differences and similarities.

## Chapter's Contents

## 2.1 History

One of the best ways to understand something is to know where it comes from. The history of scientific computing can be date back to 1945 with the Von Neumann large scale computing machine design [Nas90]. But, in the following, we are focusing on the modern era of both High-Performance Computing and Data Processing fields.

### 2.1.1 History of High-Performance Computing

The first Commercial Supercomputer was designed in the 1960s, by Seymour Cray (the founder of the Cray company) and by IBM, to solve scientific problems. At the same time, the scientific field of High-Performance Computing emerges from the need to leverage performances from these new machines.

In the 1970s, the first benchmarks start to appear in the HPC community, and among them the famous LINPACK [Don+79]. In this period, supercomputer architecture is based on vector processors that permit to apply one single instruction to multiple data stored in vector or array structure. Cray starts to introduce limited computation parallelism.

In 1980, a technical report of the United State Department of Energy, states that there are probably no more than 100 HPC systems in the world [Buz+80]. This report also emphasizes one of the key issues in HPC: **software needs to be adapted to machines' architecture in order to get close to the maximum hardware**

**performances.** Also, the vector processors start to show their limits. The main problems of this kind of architecture are the cost of those specific processors and the fact that the gain of performance is limited to some specific data-intensive workload. It is in the mid-'80s that parallel computing starts to emerge [Rat85]. The new problem the developers are facing is the interconnection between the growing number of processors. Accordingly, the network becomes critical to get performance out of this new kind of architecture, and the `interconnect` field has risen to create very low latency network between computing units.

In the 1990s, a new LINPACK benchmark implementation called HPL is used to sort the 500 fastest supercomputers by performance. This ranking is called the top500. It is still updated twice a year and gathers a lot of information on the most powerful computer in the world [Top18].

In the same time, the multiplication of expensive workstations gives the idea to the Berkley university to group them in a cluster in order to do computation out of the work hours [ACP95]. Then, the personal computers (PC) starts to emerge and the Beowulf project leverage nascent standard, Linux operating system, and Ethernet, to create the first commodity clusters [Ste03].

Commodity CPUs became more and more common from the year 2000, because of the lower cost and the ease of programming due to simpler architecture. Also, graphical processing units (GPU) are starting to be used in HPC because of their great vectorization capacity that accelerates programs the same way that vector processors use to do. Nowadays, this architecture is ubiquitous and the differences between the clusters are mostly based on the network topology. Indeed, as the number of computation nodes increases the network topology became more large and complex.

The storage issue crosses the whole history of HPC. Most of the HPC systems have a centralized Input/Output (I/O) system that serves application data requests to a controller that schedule I/O thought multiple hard disk drives (HDD). The file system is distributed over those disks where it can do each I/O operation in parallel, e.g. read from several disks one piece of data. The first I/O system approach that implements this mechanism is the redundant array of inexpressive disks (RAID) [KGP89]. These techniques increase the scalability of the file system by summing multiple disk bandwidth and avoiding failure problem though error codes. Similar approaches, but based on a file system distributed over multiple machines, like Lustre or GPFS, are now widespread in the Top500.

### 2.1.2 History of Data Processing and Big Data

Big Data history is much more recent. It coincides with the rise of the Internet and was invented by the biggest Internet companies to cope with the huge amount of data they are facing with the massive adoption on the Internet. But, the kind of problem that the Big Data is facing, big I/O throughput on large storage facilities, was already present in the industry with the mainframe computer since the early 1960s. This kind of computer was also used in the scientific world for all data-intensive workloads. With the emergence of the Internet in the 90s, the scientific communities are now capable of connecting the clusters together in order to process larger amount of data. This field is called the High Throughput Computing [Liv+97] and relies on the Grid [Nas90] infrastructure.

The term `Big Data` first appear in the scientific literature in 1997 with this definition: The Big Data problem is when a single dataset does not fit into one computer's memory or disk [CE97]. More generally, Big Data concerns the whole data pipeline in a distributed system: ingestion, storage, formatting, cleaning, processing, querying, and curation.

The Big Data scientific field is driven by the Web industry. The foundational papers of this field of research come from Google. Two main aspects are being addressed, the distributed file system with the Google File System paper in 2003 [GGL03], and the data processing framework with the Google MapReduce paper in 2004 [DG04]. The problem is that Google does not open-sourced these tools, so the Apache foundation starts funding a crawler to index the Web called Nutch in 2005. From this project emerges the needs of a distributed file system and data processing framework, so they made new open source tools based on the Google's papers, these two projects moved to a new entity called Hadoop. In 2006, Doug Cutting, one of the founders of Hadoop, was hired by Yahoo to work on this project and it starts to run it in production the same year.

In 2009 is founded Cloudera, the first company that proposes a commercial Hadoop distribution. The next years, Facebook, that embraces Hadoop for some time, is claiming that they own the largest Hadoop cluster in the world with 21 PetaBytes (PB) of storage. Another company takes shape in 2011, HortonWorks which is a Yahoo spin-off. In 2012, Facebook goes to 100 PB of storage, increasing by 0.5 PB per day.

Hadoop 2.0 is released in 2013 with a new major component, YARN [Vav+13b] which is a resource and job management system (RJMS). This new Hadoop entity becomes necessary to share resources between all the new frameworks that appear in the Big Data community, and also share Hadoop clusters between different organizations. In 2017 Hadoop 3.0 came out with a lot of new features and optimizations but no major change.

Since 2007, Hadoop and the Apache Big Data ecosystem, become the de facto industry standard and still is with more and more contributors from large companies: Facebook, LinkedIn, eBay, Twitter, Amazon, Apple, IBM. . .

In the meantime, the CERN is developing its own storage capacity with 20PB stored on tape in 2010, 100 PB in 2015. It reaches 200 PB in 2017, and is now around 320 PB [CER19].

### 2.1.3 History of the convergence

In the Big Data field, the main constraint is to scale on unreliable commodity hardware. It should run on any kind of machines that can be switch dynamically in case of failure or upgrade. So, Big Data tools are made for resilience and scalability, but not for performance. Those tools are not related to the hardware, this is even one of the requirements. This was not a problem because most of the workloads in data processing underutilize the cluster computation capabilities.

However, we recently witness the emergence of the Machine Learning in all its forms. This new kind of workloads is computation intensive. Thus, the Big Data community and the high performance computing community start to converge.

At the resource management point of view, in 2011, a tool called Hadoop on Demand (HOD) is released with the Hadoop 0.22 version. It uses the HPC RJMS Torque to create a Hadoop cluster on demand on an existing HPC cluster. But each HOD deployment is limited to one user, and does not provide the deployment of the Hadoop distributed file system (HDFS). To overcome these limitations, the University of San Diego proposes to import Hadoop on their HPC cluster using a simple install script to be integrated to the RJMS job script, in order to run Hadoop inside a user's HPC job [KTB11]. In 2013, another approach called Job Uninterrupted Maneuverable MapReduce Platform (JUMMP) is doing a finer grain integration between the HPC RJMS and Hadoop by creating a job for each Hadoop task [Moo+13]. To be able to run arbitrarily long Hadoop job, it leverages the MapReduce dynamicity by moving regularly some tasks from node to node with the HDFS data attached to it. More tight integration is proposed using the grid oriented Pilot abstraction. This project started in 2013 and was published in 2016 with a complete evaluation carried out on a classical HPC and a converged machine [Luc+16].

At the runtime level, the Network-Based Computing Laboratory of the Ohio State University has started an initiative called High-Performance Big Data (HiBD) in 2010 [Sur+10]. They are modifying the communication layer of standard Big Data tools (Hadoop, Spark, MemCache, and HBase), with the HPC network technology called remote direct memory access (RDMA) over the Infiniband interconnect.

Since 2014, The High-Performance Computing Enhanced Apache Big Data Stack (Harp) [ZRQ15] project is also working on a similar approach, e.i. They also change the network implementation but instead of using direct HPC technologies they choose to implement the HPC concepts into Hadoop: they replace the point-to-point communications of Hadoop by a collective communications scheme based on the MPI model. Also, the same team connects Hadoop to the Intel Data Analytics Acceleration Library (DAAL) under Harp in order to speed up the data analytics computation [Uni18].

More generally both communities are looking at the other. Evidence of this convergence is the fact that papers from conferences of both fields are talking of each other. For example, at least 6 papers from the IEEE BigData 2016 conference are about high performance, and at least 9 papers in IEEE Cluster 2016, which is an HPC oriented conference, refer to Big Data[1]. Furthermore, the HPC-Big Data convergence was the focus of IEEE Cluster 2017.

## 2.2 Workloads

What really defines those fields in the first place is the workload they are designed to cope with. The main difference between the two fields is that HPC workloads are computation centric and I/O latency sensitive, but Big Data workload is data-centric and I/O bandwidth sensitive.

[1]This was calculated by extracting keywords from the papers' titles

Comparing the two kinds of workloads, multiple studies highlight that Cloud traces released by Google are very different from HPC and Grid traces. The differences concern the shape and the submission rate of the jobs but also resource usage and the kind of failure [DKC12; Amv+18].

The following is a detail view of each type of workloads.

## 2.2.1 HPC workloads

HPC workloads are mostly composed of parallel applications that run on a very large number of computing unit in parallel with frequent synchronization. To handle this efficiently, a dominant share of the HPC applications are based on the Message Passing Interface (MPI). It provides communication primitives to send and receive messages, either synchronously or asynchronously, with point-to-point communication and more complex communication patterns called collectives (e.g. gather, scatter, broadcast, reduce, ...). This standard has many implementations, more or less compatible with each other, that are developed by the HPC community (e.g OpenMPI, MPICH) or provided by vendors like (e.g. IntelMPI). The reference implementations are coded in C, highly optimized to minimize message latency, which is critical for performances of parallel applications. Most of the MPI implementations leverage very low latency Network Interface Controller (NIC) called "interconnect" that are common on HPC clusters. Part of the MPI work may be offloaded to specific hardware functions that are part of these NICs.

At the resource management level, the Parallel Workload Archive [Fei17] makes publicly available a list of HPC workloads. These traces are extremely valuable for the HPC scheduling field of research because they contain all the resource requests triggered by HPC users and how the actual scheduler did answer to these requests. It also contains some information on the jobs like the execution time and in some traces the average usage in memory and CPU. Dror Feitelson, which host the workload archive has also written a book on the modeling of HPC workload that extracts models from statistical studies of real workloads [Fei15a]. But, these traces are taken from the RJMS point of view and thus, the jobs are seen as black-boxes, e.i. traces do not contain any details on the job itself. Indeed, probably due to security and privacy issues, a complete HPC workload trace with detail job traces remain unavailable for now.

In fact, the types of HPC applications that are running inside the jobs are numerous. In order to represent all types of applications, the HPC community is traditionally using lots of different benchmarks, and more complex applications called “mini-apps” or “proxy-apps” [ICL18; NAS16; OAK18; Ner18]. A more recent initiative called the Exascale Project provides a list of modern proxy applications: the Exascale Computing Project (ECP) proxy applications [Pro18].

### 2.2.2 Big Data workloads

In the Big Data field, workloads mostly consist of task-based applications for indexing, processing, or querying big amounts of data. The Big Data itself is often characterized by these three properties: volume, variety, and velocity. It means that the datasets are big in volume, contain different kinds of structured and unstructured data that comes from a large variety of sources, and the pace at which the data is coming to the system can be very fast. These statements have multiple implications on the actual Big Data workload. It is worth noting that when we describe a workload, a big part of the Big Data issues are avoided; i.e. we only describe the processing part and not the whole data management workload that is hidden in the infrastructure.

The reference paper on the Big Data workloads from Chen et al. [CAK12] explores 7 different production traces from multiple companies including Facebook, for a cumulative time of approximately one year of Hadoop MapReduce traces. Even if the study starts to age now, because the traces was produced from 2009 to 2011, it gives a good overview of what a Big Data workload is. Insight on how these kinds of workloads are evolving is given in this paper with the two Facebook traces that are respectively from 2009, and from 2010: we can observe a huge growth in the amount of data moved, and on the input size that goes respectively from 1.5 PB to 1EB per month.

Most of the workloads are very different but some characteristics are common to each of them:

- Small jobs (less than 1min) dominate all workloads in term of number of jobs: but small jobs input sizes go from KB to GB for some workloads.

- 80% of the data access goes to less than 8% of the data.

- There is a temporal locality of data access with 75% of already accessed data are re-accessed within 6 hours.

- Skew in access frequency is observed on every trace with the same slope: a few files account for a very high number of accesses.

- The analysis reveals bursty submission patterns across all the workloads.

One of the conclusions of this paper [CAK12] is also that Big Data is not only batch data processing, but also queries, and more recently, stream processing, which becomes more and more present in the workloads.

These remarks give an overview, but there is great variability between workloads. Thus, the authors of this paper are warning the reader that there is no “typical” workload.

Another study was released approximately at the same time by Carnegie Melon University. The focus of this paper is on the study of academic research the MapReduce workload on three Hadoop clusters [Ren+13]. The findings of this paper are very similar to the one of Chen et. al., and the previously mentioned characteristics are holding. Also, this paper provides a study of the configuration changes and user’s job customization: it exposes that Hadoop is not dealing with the diversity of application correctly and that default configuration leads to unbalance load on the cluster. It also shows that diversity exists not only between workloads but also inside each workload.

Big Data applications that composed these workloads are also a very wide mix. To understand what’s happening inside the applications, each of them can be split again into smaller computation units. Gao et al. [Gao+18] define eight classes of computation. These classes are defined according to the inputs data structure and size and the storage and network I/O patterns. They call these classes “Data Dwarfs”. These dwarfs are listed as follows: Matrix, Sampling, Logic, Transform, Set, Graph, Sort, and Statistics. A Big Data workload is a mix of applications that are themselves composed of a variety of computation units. The part of each application type in the workload mix highly depends on the Big Data cluster’s users’ needs, which partly explain the workload variability.

Because of Hadoop’s tight integration between the resource management and the applications, Big Data cluster traces contain much more details on the applications than in HPC, where applications are seen as black boxes. Moreover, and probably because of the data culture of the people that are building these tools, the Big Data runtimes are giving a lot of information on their internal behavior. For example, Spark is providing a whole event base detailed trace for each application that can be used for example for performance analysis.

### 2.2.3 Workloads Convergence

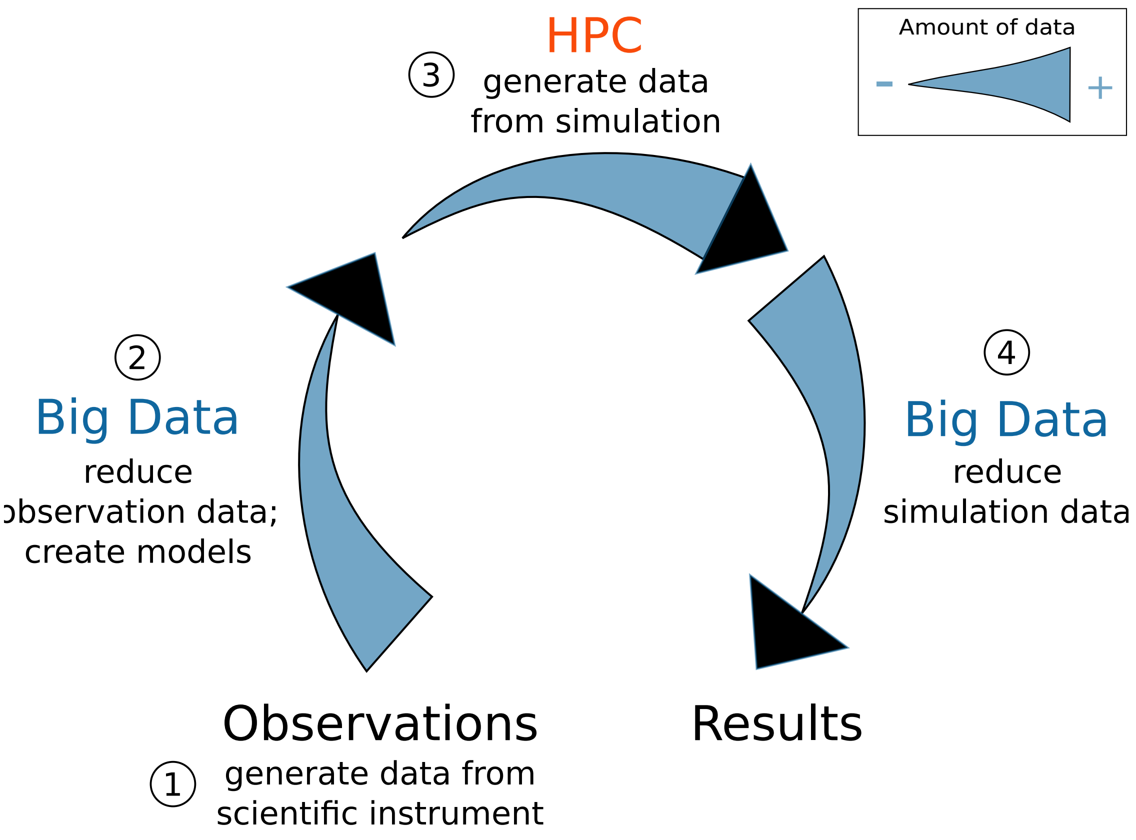
From the Big Data industry point of view, using HPC is attractive because it can speed up the computation intensive part of their workloads. The most prominent example is Machine Learning workloads and especially Deep Learning. Indeed, Deep Learning, which is based on large neural networks with more than a hundred layers, executes a large number of matrix multiplications for training their models (Billions of FLOPS) [He+15]. This model training is mostly done on HPC clusters or dedicated clusters with GPU. A good indicator of this convergence is that three different deep learning tutorials have been held at the SuperComputing 2018 HPC conference.

From the HPC scientific community point of view, the workload convergence emerges from two factors. First, while supercomputers become larger, scientific simulations become more and more precise and the size of their output is growing accordingly. The second factor is the increase the precision of all the sensors that capture real-world behavior, for example, the Large Hadron Collider (LHC) is the world's largest particle collider has generated around 50 PetaBytes of data in 2018 [CER18]. The Big Data and Exascale Computing (BDEC) project recent report (2018) states that the amount of data produced by several scientific instruments exceeds the storage capacity of the current HPC centers [Asc+18].

Figure 2.1 is presenting the scientific workflow that involves HPC and Big Data. This workflow includes multiple steps that are detailed here:

1. First, data is generated from real world observation through scientific instruments and sensors.

2. This amount of data needs to be reduced to extract useful information from it through Big Data methods and tools.



HPC
3 generate data from simulation
Amount of data
-
+
2 Big Data
reduce observation data; create models
4 Big Data
reduce simulation data
Observations
1 generate data from scientific instrument
Results

Figure 2.1.: Scientific workflow that includes Big Data and HPC: 1. Lots of data is generated from observation of the real world, 2. It is reduced to generate a simulation model, 3. Simulation of this model produces large amounts of data, 4. The simulation data is reduced to extract scientific results. This graph is highly inspired from Figure 4 of the 2018 BDEC's report [Asc+18].

3. From this useful extracted data, the scientists are creating a model in order to run a simulation.

4. The simulation produces large amounts of data that needs to be processed to extract useful pieces of information that leads to scientific results.

In this workflow, steps 2 and 4 involve Big Data but the implementation of each step can be completely different. Indeed, the Grid community is working on step 2 since decades, i.e. managing big amounts of data produced by big scientific instruments like particle colliders. In fact, because of the needs of interconnection of computing centers and great physics instruments, they become the pioneer in the network technologies: the World Wide Web was invented in the early 90s at the CERN, which is the largest nuclear Physics' laboratory in the world [Wik18].

HPC community is already facing the problem of step 4, i.e. managing data produced by large simulation. A whole scientific topic called “in situ” is addressing this problem. In situ, means that produced data is pre-processed directly on site, at the output of the simulation, to produce for example real-time visualization [Ma09].

So, if all the parts of this workflow are already under study, what are the real challenges of the HPC and Big Data convergence? The real issue is to find a way to execute a whole workflow on a unified infrastructure. For now, a scientist that has this workflow need to interact with many systems and to move data around manually, which is error-prone, time-consuming and not efficient in term of resource usage. This unification could be made at different levels: at the hardware level with high-performance computer boosted to cope with large I/O requests, or at the software infrastructure level with the integration of this workflow into all the infrastructure stack (user portals, resource management, file system, application runtimes, etc.). Both are explored in the following sections.

## 2.3 Applications and Runtimes

An important part of both HPC and Big Data software stacks is their application runtime. A runtime can be defined as a basis for the applications that implement common features related to synchronization between processes, and life cycle management. Here is a description of them and of the possible points of convergence.

The software stack that is standard in Big Data is mostly based on portable binary code, or bytecode, executed on a Java Virtual Machine, and the level of abstraction proposed to the users of those runtimes is high. The first model proposed by Google in 2004 was the MapReduce programming model. Inspired by functional programming concepts, this model allows the users to give a series of functions to be applied on data without taking care of parallelization or data movement [DG08]. Most of the runtimes, also known as frameworks, that were developed afterward, derived from this model. Also, those frameworks are made to be tolerant to failure, which implies an overhead due to the mechanisms that provide resiliency. For MapReduce for example, each task is independent and if any of them is failing, it is just restarted elsewhere. The culprit of this mechanic is that intermediate data cannot be kept in memory, which leads to a lot of data movement. A more recent framework (2012), called Spark, developed the concept of resilient distributed datasets (RDD) to keep everything in memory. It leverages lazy evaluation of the data transformation to achieve resiliency [Zah+12]. Spark became a standard with more than 60 contributors from 25 different companies and 5 different Universities [Fon18].

Runtimes in HPC are based on a much more low level of abstraction. The Message Passing Interface (MPI) standard is based on messages that achieve point-to-point and collective communication between processes. It is widely used as a base layer to build large parallel HPC applications. The Open Multi-Processing (OpenMP) application programming interface (API) permits applications to speedup intra-node parallelization computation because the inter-process coordination is done inside, using shared memory. It enables semi-automatic parallelization of C/C++ and Fortran code using compilation extensions. OpenMP, that is managing intra-node coordination, is often used in conjunction with MPI for inter-node coordination, to achieve the best of both worlds. To handle the heterogeneity brought by cluster accelerators it exists another framework called OpenCL. The programming model of OpenCL gives to the users an abstraction that encompasses all the different kinds of computing devices and thus permits to code one computation kernel (units of computation logic) that will be optimized depending on the hardware it runs on (if it is supported by the framework). But, most of the accelerators nowadays are Nvidia General-Purpose computing on Graphics Processing Units (GPGPU), and better performance are obtained by using the direct programming interface proposed by Nvidia: Compute Unified Device Architecture (CUDA). Another standard developed by Cray called Open Accelerators (OpenACC) emerges recently (2012) to enable programming on heterogeneous CPU/GPU resources. Similarly, to OpenMP, it is based on compiler's preprocessing directives that are placed by the users in their code. The code is then parallelized on GPU by the OpenACC. In fact, OpenMP also supports GPU since 2013 with the 4.0 version of its specification.

HPC applications also employ lots of very optimized libraries for generic computation patterns. Some very common mathematical operations, like vector and matrix simple manipulations, are specified by the Basic Linear Algebra Subprograms (BLAS). It exists several BLAS compatible implementations, in multiple languages, that support most of the computing unit's architectures. Based on these implementations, other higher-level libraries (e.g. Linear Algebra PACKage (LAPACK)) are providing more complex operations like linear equations solvers or matrix factorizations. Some vendors are also distributing their own implementation hand-optimized for their own hardware. A famous example is The Intel Math Kernel Library (MKL) which also implements fast Fourier transforms (FFT) and vector math. The first point of convergence is located at this level, with the Intel

Data Analytics Acceleration Library (DAAL). This library is designed to optimize classical data processing and machine learning operations on Intel hardware.

But, let's take a step back to define the different convergence points between HPC and Big Data runtimes. The first question that arises is: do the Big Data runtimes have to be kept as-is, modified, or even completely rewritten, to be able to get the maximal performance proposed by HPC hardware? In fact, the three approaches exist and are detailed in the following sections.

### 2.3.1 Keep as-is

Keeping an unmodified Big Data stack has one main advantage: keeping the community version means have regular performance upgrades and new features for free. In contrast, a modified stack means that you have to sustain the cost of maintaining a custom software that needs to be updated with the master branch, and this cost increase with tools that are evolving fast (which is the case for Spark with 2598 commit on the master branch in 2018).

Chaimov et al. studied the scaling capability of Spark on HPC systems [Cha+16]. This paper reveals that on large Cray supercomputer Cori and Edison, the scaling of Spark is mostly limited by the Lustre parallel file system metadata operation latency. The approach that is retained in this study is the use of a local file system, backed by Lustre, is used to store application data. With better performance than using Burst Buffers and similar performance than an in-memory file system, this approach is retained because it is more reliable than an in-memory file system that crashes the node when full. With the addition of a caching layer for file system metadata, this approach permits to Spark to scale to up to 10000 cores. But, this scalability capacity is not accompanied by scalability efficiency number.

Another study is continuing this work by doing a fair comparison between Spark and MPI with the implementation of 3 different algorithms used to explore TeraBytes (TB) scale scientific datasets [Git+16]. In this paper, they implement the same algorithms with Spark and with MPI using the BLAS library underneath for both implementations. Leveraging the Chaimov et al. experience [Cha+16], they manage to scale a Spark application up to 1522 nodes (48704 cores), but at this scale the time spent in the task scheduling (scheduling delay, straggler effect, and task overheads) becomes dominant: Spark's scheduling needs 10x more time than the application itself. Also, the MPI implementation runs 26x faster than the Spark one at this scale but this goes down to 2.3x for specific applications, with a mean around 10x for the 7 experiments proposed. These results are corroborated by other similar studies [ROA15a; Dün+16].

There is a current effort to improve Spark scheduling performance and predictability by separating the scheduling for each type of resource [Ous+15]. The performance gain obtained in the order 10%, which is not comparable to the gap of performance between unmodified Spark and hand-optimized code.

Keeping the runtimes as-is seems to be a good option to pool the effort of both communities. But this approach is limited because at a certain point improving performance requires radical structural changes, e.g. replacing the whole network stack of the runtime.

### 2.3.2 Modify

Another approach is to keep the user interface while adapting the runtime to high performance hardware. This approach has the advantage to be transparent to the Big Data end users.

ARION [Xu+17] It is made to automatically infer the best acceleration techniques for matrix multiplication. Its strength is that it works for unmodified Spark applications. To do so, this approach is examining the matrix size and properties (i.e. sparse or dense) and delegate the computation to the best available implementation using different CPU and GPU BLAS libraries.

This approach is appealing because modifying only the core of the runtime permits getting better performances while keeping the user's interface. But, this approach has limitations, because the most efficient implementation may require a lot of effort to be derived from the framework programming model exposed to the users if it is even possible.

The approach of Harp [ZRQ15] go a step further by changing the programming model, and consequently breaks the user interface. Harp is a collective communication plugin for Hadoop that replaces the Map-Reduce model by a so-called "Map-Collective" model. This mechanism opens the possibilities to lots of optimization, but it arms resiliency that is only achieved at job level and not at task-level like it is done in MapReduce.

With some benchmarks implementations using Harp collectives, the paper shows an almost linear strong scalability (increase the number of nodes for a fixed problem size) on the Big Rad II supercomputer up to 4096 cores. Still, there is no performance comparison with other implementations, so even if it scales correctly it is impossible to compare with an implementation that would use HPC technologies.

However, another paper is providing this kind of comparison for a specific algorithm called Latent Dirichlet Allocation (LDA) [Pen+17]. Using Harp collectives on Hadoop, the HarpLDA+ implementation is comparable or even outperforms state-of-the-art implementations that are written with MPI. This is mainly due to a better algorithm implementation which minimizes the cost of synchronization but shows that Java-based implementation is capable to compete with lower-level compiled languages — at least in some specific cases.

To improve Harp performances without rewriting all the computation kernels, another approach is to use the implementation provided by Intel DAAL [Int18a]. Harp-DAAL [Uni18] is a prototype which implements this idea, but no performance numbers have been published yet (in 2019). DAAL is advertised by Intel to be from 4x to 200x faster than Spark, it will be interesting to see if this speedup is cumulative with the one provided by Harp itself.

### 2.3.3 Rewrite

A more radical approach is to rewrite the entire Big Data analytics tools with HPC technologies in order to fully leverage hardware capabilities. The problem is that rewriting the tools entirely means changing the user interface, which is a severe issue. Indeed, one of the main strength of the traditional Big Data analytics frameworks is to provide a simple interface to the users in languages that are already widely used in the data science community. With this constraint in mind, Intel developed a new tool called High-Performance Analytics Toolkit (HPAT) that provides a high-level scripting language interface (in Julia, and now in Python [Int18b]), that is compiled down to C++/MPI code highly optimized through multiple steps of compilations [TAS17].

This last approach of rewriting the whole runtime may be able to get the peak performance out of the HPC hardware, but it will very hard to drag users to this kind of tools unless they bring disruptive advantages.

In conclusion, whether the Big Data application needs to keep, modify, or rewrite, is still an open question. I think that the 3 different approaches will live together for some time because using one or another really depends on the trade-off between ease of use and performance for the end users, and the trade-off between grouping efforts and specialization for the maintainers.

## 2.4 Hardware

As mentioned in the history section above (2.1), the Big Data and HPC background are very different. That's why the underlying hardware that supports their respective infrastructures is different. This Section presents the hardware characteristics associated with each domain.

### 2.4.1 Big Data Hardware

In the case of Big Data, the hardware used is commodity "off-the-shelf" servers. The choice of this kind of hardware is primarily due to cost reduction, but also because data-intensive workload requirements are less demanding than computation intensive workload: the data movement between network, disk, and memory is the performance bottleneck in most of the applications [Mar+16].

Because Big Data a cluster is composed of commodity hardware, the network used to connect them is a standard Ethernet network.

The Big Data infrastructure is driven by the need to store massive amounts of data. In consequence, Big Data distributed file systems demand a lot of storage capacity, and to cope with hardware failure and minimize data movement, the data is distributed on every node of the cluster. That's why all servers are equipped with cheap and large capacity disks. This architecture has the advantage to scale the file system capacity and the computation capacity at the same time when a new machine is added to the cluster.

So, a classic Big Data infrastructure is a set of commodity servers with standard Ethernet network cards and large disks.

### 2.4.2 HPC Hardware

In the HPC side, supercomputers are made to be efficient in terms of computation. Most of the HPC workloads is computation intensive large parallel application that runs on hundreds to thousands of CPUs simultaneously. To be able to synchronize between computation processes, applications are sending messages and tries to minimize the time they wait from the network.

These characteristics make them very sensitive to message latency. It means that each node should be as big as possible in terms of numbers of computation cores because all the processes that are on the same node can communicate very efficiently. Also, the communication between nodes should have the slightest possible latency. To achieve this, all the components contained in supercomputers are at the high end of the state-of-the-art: processors with a large number of cores and vectorization mechanism (e.g. Intel Xeon), large high bandwidth memory, and low latency network interconnect, e.g. Infiniband which was used on 77% of the top500 clusters in 2017 [Fon17], and the more recent Atos BXI [Der+15] and Intel OmniPath [Bir+15]). The use of so-called "accelerators" is also common in HPC. Accelerators are mostly graphical processing units (GPU) that are very effective with vectorized computation, but some field-programmable gate arrays (FPGA) can also be used because they are very power efficient to achieve some specific tasks.

Unlike Big Data clusters, the HPC nodes do not contain large HDD. They either have no disk at all (this is called "diskless" architecture), or a small SSD used by applications to store intermediate results (called the "scratch" space). Some recent clusters are now shipped with Non-volatile memories (NVM) that is mostly used to replace the SSD for the scratch space. Data is stored in dedicated storage cluster. Each nodes of this cluster have multiple disks that can be accessed in parallel. This storage cluster is attached to the computation cluster with a high performance interconnect.

It is worth noticing that, in order to get the peak performance HPC hardware and software are tightly coupled. For example, some interconnect are aware that MPI collective communications are used by the software and try to offload part of the message processing directly on hardware.

### 2.4.3 Converged Hardware

More and more institutes are purchasing high-performance data analytics machine. The Wrangler cluster [Gaf+14a] was built by the Texas Advanced Computing Center (TACC), to support data-intensive research. They use HPC technologies, like the Lustre parallel file system over Infiniband, but with better connectivity than the usual. Wrangler also features a NAND flash storage that permits reaching 1TB/s of bandwidth (6x more than an SSD and 40x more than an HDD). It also has the capability to ingest and export data from the outside world with a 100Gb/s Ethernet connectivity.

This kind of converged hardware is accompanied by a software infrastructure that enables the users to use both HPC and Big Data traditional software stack, but not a converged way of managing the resources, or a high-performance data analytics runtime [TAC18].

## 2.5 Infrastructure

We can define the infrastructure as the set of shared services that are provided to the cluster as a whole. The following sections are describing the different part of this infrastructure for HPC and Big Data clusters.

### 2.5.1 Resource and Job Management System

The resource and job management system is the core component of a cluster: It controls the resource sharing between users, implements the scheduling policy, manages the cluster energy and manages the users' jobs life-cycle.

Recently, several efforts are made to make HPC and Big Data work on the same platform using different approaches. This section discusses these approaches and explained the remaining challenges still to be taken on about resource management convergence.

Traditionally, jobs that run on the HPC environment are Physical simulations using parallel computation frameworks to parallelize the work across multiple computing units inside a node like OpenMP, or between the nodes like MPI. These tools' architectures are made to scale vertically (scale-up) so they make extensive use of each node memory, CPU and GPU computation capabilities.

HPC clusters are managed by a Resources and Jobs Management System (RJMS). Georgiou's Thesis propose a complete study of HPC RJMS [Geo10]. The RJMS allows multiple users to run several parallel jobs simultaneously. To do so, it maintains a global cluster state and allocates jobs on nodes (or part of nodes: cores) by using complex multi-criteria scheduling algorithm. The jobs have a walltime. It means that the user that submits a computation job also submits an upper bound in time (or walltime) that enables the RJMS to schedule the job in the future if necessary. This mechanism achieves really good utilization rate on a crowded cluster, but it does not allow the jobs to dynamically scale up or down: a job must acquire the totality of the necessary resources for all its run time (this put the cluster utilization in perspective. . . ).

HPC hardware is made to be robust: redundant supply, redundant network, error-correcting code memory (ECC memory), . . . Because hardware faults are rare, the resource manager is usually not dealing with them automatically.

Also, the HPC traditional environment is compute-centric: It usually does not have to manage a lot of data and is focus on computation power and efficiency so the RJMS do not communicate with the file system, which is centralized on a dedicated set of nodes.

In the Big Data world, the first class citizen is obviously data. Thus, the management tools are data-centric: the data locality is more important than the computation.

Most of the Big Data tools, like the Hadoop project, are Open Source and supported by the Apache Foundation [Fon19b]. They are usually deployed in a complex stack with, in the case of Hadoop, HDFS as a file system, YARN as a resource scheduler, MapReduce as an application framework, and the user application on top of it. Some additional tools like Pig or Hive are built on top of MapReduce to query the resulting data efficiently.

All this software stack is designed to run on unreliable hardware infrastructure, to be fault-tolerant, and to scale horizontally (scale-out). So each piece of software in integrated and communicates to dynamically adapt to hardware failure. Hence, Big

| | HPC cluster | Big Data cluster |
|---|---|---|
| Goal | Maximize resource reservation | Maximize resource utilization with spare |
| Scaling | Scale-up and scale-out | Only scale-out |
| Resilience | Hardware | Software |
| Resource allocation | Static | Dynamic |
| File System | On dedicated storage nodes | On compute nodes |
| Locality-aware scheduling | No | Yes |

Table 2.1.: Summary of the differences between HPC and Big Data platforms.

Data RJMS is made to detect node failure and support dynamic resource allocation even during the jobs' execution time.

Big Data file systems use the local storage on each node to is order to scale linearly in term of capacity, and use replication for fault tolerance and load balancing. Indeed, the RJMS is capable to exploit the data locality by taking resource affinity request into account, thus avoiding data movement and save network bandwidth.

These two descriptions of the Big Data and the HPC resource management clearly expose the deep design deferences between them: HPC RJMS are made for reliable hardware and are more efficient for computation workloads thanks to deadlined jobs, and sophisticated scheduling algorithm; the Big Data RJMS are capable to handle node failure and uses data locality-aware which is critical considering that the data movement is the bottleneck for this kind of workload.

Also, the objectives of both systems are different. For example, regarding the cluster utilization, it has a different meaning in each world:

- For HPC, the target is 100% *reserved* resources on the cluster through users job submission.
- For Web cluster scheduler, the target is to maximize actual resource (CPU, Memory) utilization but to keep spare resources to ensure SLA in case of failure.

Big Data and HPC RJMSs are very different and specialized for their workloads. They both fail to handle the workloads of the other properly. Our goal is to find a solution that performs just as well on computation centric and on data-centric workloads. To do so, we look forward to getting the best of both worlds, HPC scheduling performance, and Big Data data awareness and flexibility, into one optimized solution. Chapter 4 is providing a detailed study of the different kind of RJMS collaboration techniques followed by our approach description, along with an experimental evaluation.

## 2.5.2 File Systems

In large computer system infrastructures, data has to be efficiently accessible from anywhere and securely replicated to be fault-tolerant. This is the job of the cluster file systems. Lots of file systems exist and each of them is trying to meet the requirements of the workload they were made to deal with.

#### Big Data File Systems

In the Big Data field, the file system is a key factor in the system overall performance: the main performance metrics are read and write bandwidth and file access time (also called latency). As the file system is shared between all the applications of the cluster, these metrics need to be optimized while avoiding network and disk contention, but also while taking into account fair sharing between applications. It also has to deal with a massive amount of data continuously injected in the system, e.g. back in 2012, Facebook claims that they were already storing 100 PB of data, increasing by 0.5 PB per day.

To cope with these constraints, Big Data file systems are designed to be scalable, resilient, and optimized to read and write large files. To do so, the data is split into large blocks that are distributed across all the nodes of the cluster using each of the node's local disk as part of the global storage. Each block is replicated, generally 3 times, in a different part of the cluster to ensure data availability in case of failure of disk, server, or even rack of servers. For example, HDFS is writing once locally, once in a random node on the same rack, and once in a random node in another rack.

A centralized service is serving metadata to the file system clients. It is responsible to maintain the association between the data namespace (or paths) and the data blocks, and a map where the blocks and their replica are. This metadata file system server also triggers the replication mechanism in case of failure, to maintain a constant number of replica for each block. The metadata server itself has a failover server in case of failure.

One key feature of the Big Data file systems is the exposure of the locality of data blocks. Thanks to this feature, the work can be placed close to the data in order to maximize the bandwidth, minimize the latency, and minimize the data movement. This data-aware scheduling policy allows giving to the applications the resources that host the data they needed to access, and thus this access is done directly on the local disk. This mechanism relies on the collaboration between 3 components: the Resource and Job Management System (RJMS), the Big Data Analytics Framework (BDAF), and the distributed file system (DFS). The following is an explanation of the locality-aware placement process:

1. A user submits an application to the BDAF, associated with a resource request.
2. The BDAF processes the application to get the data paths requested by the application in the DFS.
3. The BDAF asks the DFS metadata server for the input data blocks location.
4. The BDAF makes a resource request to the RJMS based on the application resource request, and it adds locality preference based on the information collected from the previous step.
5. The RJMS tries to fulfill the locality preferences by waiting for the requested resources to be available, for a certain amount of time.
6. Then the RJMS choose a set of resources and allocates them to the application.
7. The BDAF can now start the application on the allocated resources.

This description is based on what is implemented in Hadoop with YARN as RJMS. An alternative approach, called Mesos, works differently for steps 4, 5 and 6. Indeed, instead of taking resource requests, Mesos is making resources propositions to BDAF. Thus, the BDAF is able to achieve his goal of locality by selecting the wanted resources from the resources propositions [Hin+11].

It is worth noticing that the replication of the data blocks over multiple resources on the cluster is not only useful for resiliency but also to achieve data locality: The more there are replicas, the more there are resources that fulfill locality constraints. But replication has a cost: each block written to the file system are stored 3 times by default. It means that a DFS storage capacity is only $1/3$ of its cumulative disk capacity.

Finally, in order to improve data streaming and data staging, the POSIX standard file system requirement are relaxed by most of the Big Data file systems. The file system also exposes locality and replication parameters to the applications. It means that a specific client is needed to access to the file system because OS tools are not compliant with the file system API.

### HPC file systems

HPC file systems are built with different constraints in mind. On an HPC cluster, the file system is mostly used by applications to write large checkpoints regularly during their execution, e.g. every 100 steps of simulation. From the file system point of view, these checkpoints are big bursts of data that need to be absorbed as fast as possible. If the file system is not capable to sustain the pace of I/O writes, the applications would wait and thus waste computing resources. To manage the I/O bursts, the HPC file systems are reading and writing in parallel thanks to data striping between storage nodes. Each node also uses RAID technology on local disks, which allows reading in parallel on multiple disks while providing multiple disks fault-tolerance [Wik19a]. That's why HPC file systems are often called parallel file systems (PFS).

Compared to DFS, the resilience system of the PFS is much more efficient in terms of storage: it typically cost only 35% more raw storage space, instead of 300% for the DFS [Kep+15].

Another constraint is that HPC applications should not be disturbed. Indeed, as HPC applications are highly tuned and optimized to reach the peak performance of a machine, uncontrolled access to an application's resource may disturb the application expected behavior and thus lead to a loss of performance, and a waste of resources. So, HPC file systems are designed to share nothing between applications except the file system itself. Their architecture is similar to Big Data file system, with one metadata server (and its failover) and several data servers.

The main difference between the two kinds of file systems is in the way they are deployed. A PFS is centralized in a dedicated data servers cluster that is linked to the compute node cluster through high speed interconnect. But this centralization leads to I/O contentions which arm the PFS scaling [Use+10].

The response to the bottleneck problem created by I/O burst that was found in the HPC community is to add one or more layers of aggregation between compute nodes and the file system. This layer can take several forms. The simplest one is called I/O nodes: some nodes are designated as I/O gateway to PFS. Applications are now sending data to the I/O nodes, and not to the PFS directly. It gives to the I/O nodes a possibility to aggregate data and coordinates their operation to avoid contention on the main file system. Another approach is to use Burst Buffers. Burst Buffers are large non-volatile memory storages that can be attached at different points on the topology, e.g. one per I/O node or even one per compute node. But their high cost and the fact that underlying technologies do not support too much write opens a lot of questions under active study [ABE18].

PFS are fulfilling the POSIX requirement, thus access to the file system is transparent for the applications, but a client needs to be installed on each node by the administrator. A common problem with POSIX is that a lot of metadata operations are done if the applications use a lot of small files, or make a lot of random access to the file system, because it triggers a lot of call to the metadata server and has a big impact on I/O latency [Ala+11; Vil+08].

HPC applications are aware of that problems and efforts are done towards minimizing metadata requests with file aggregation. Another approach is to do data aggregation at the I/O nodes level that coordinate to make bulk reads and writes. This survey is a good summary of all the different current approaches [Zan+18].

### File systems convergence

According to multiple studies, unmodified PFS are not suited for Big Data workloads [Mol+09; YI18; Ana+09; Xua+15]. This can be due to higher data access latency, the cost of synchronization to disks, and the lack of data prefetching and data locality exposure.

However,one study compare HDFS and GPFS and conclude that both file systems have similar performances : HDFS is better in reads, thanks to locality, but GPFS is better at writes [Fad+12]. This is true when only one application is running on the cluster but under the presence of contention on the PFS this statement does not hold.

Even on data-oriented high-performance hardware, like the Wrangler machine from the TACC, applications on top of the Lustre PFS do not perform and scale as well as using the Hadoop stack (HDFS and YARN) that leverage data locality [Luc+16].

PFS does not expose data locality information because they are designed to be deployed in a centralized dedicated set of machines. These machines are called `data nodes`, in contrast with the machines which are hosting the jobs, `the compute nodes`. Thus, data-aware scheduling is not available as-is on a classical HPC environment. An interesting approach is to deploy a PFS on the compute nodes, like in a classic DFS configuration, and modify it in order to make it compatible with Hadoop applications in order to achieve similar performance. To do so, a shim layer is added to provide an HDFS compatible interface on top of a PFS. Also, the PFS is tweaked with important optimization with the aim of matching DFS performance: data prefetching, data replication with custom layouts, and relaxed consistency. It was done on multiple PFS technologies and with different implementations: The PVFS [Tan+11] one uses Hadoop's file system API to implements its shim layer which implies operational overheads because of the modified applications stack, whether the GPFS [RDS16] implementation provides an API compatible with the HDFS client which is easier to integrate and maintain.

Another hybrid approach, proposed in [Xua+15], is to mix both technologies with two-layer storage using Tachyon in-memory distributed file system (now called Alluxio), which leverage the compute nodes memory, and a parallel file system called OrangeFS deployed on data nodes as usual. With this technique, it is possible to get a 5x speedup for read-intensive workloads with temporal locality.

A similar but more integrated approach called `Hybrid design of HDFS with Heterogeneous storage` (HHH) leverages the whole storage hierarchy of an HPC cluster along with the interconnect direct memory access capacity [Isl+15]. They redesign HDFS to implement a complete integration with the storage stack, from RAM to the Lustre PFS, in addition to new data placement heuristics and RDMA-based communication, while keeping compatibility with existing applications. They claim that their Triple-H HDFS re-implementation achieve a speedup of 7x for write and 5x for read over vanilla HDFS for the TestDFSIO benchmark, and from 17% to 40% improvement on classical benchmarks.

All the aforementioned studies are focussing on Big Data workload on HPC platforms, but what about the opposite? Are DFS suitable for HPC workloads? This proposal [RFB11] suggests that this is one of the promising architecture for the future (almost present now [Hin18]) exascale HPC. But, as explained in the previous section, HPC applications are highly tuned and a small disturbance on the HPC applications can have a dramatic impact on the overall performance.

By design, the DFS disturbs the applications with remote disk access on random nodes that are not part of the application requesting this access. This perturbation is different depending on the operations: for reads when the data locality is not meet data is fetched on another node, and for writes for the data replication mechanism triggers data transfer to some random nodes over the cluster's network. In contrast, PFS have much more predictable behavior. The only point of contention is at the data and metadata servers network or disk bandwidth. But, to the best of our knowledge, a study of the real impact of DFS on an HPC workload does not exist yet. During this thesis, we have conducted a study on this topic [Hie18]. We have shown that the impact of DFS traffic generated by a DFSIO benchmark, on NAS Parallel Benchmarks MPI applications is highly variable and can go from no impact to more than double the execution time. Unsurprisingly writes operation are more impactful than reads because it triggers much less network traffic if data locality is reached. Also, pining the HDFS client on a CPU core diminishes the impact of the DFS client on the MPI application because it does not corrupt applications' caches and can leverage hyperthreading to avoid disrupting the computation.

To go further on the understanding of file systems we need to experiment more. But, experimentation on the file system is complicated, especially when you want to compare several file systems: most of the production platform have only one file system, and setting up multiple distributed file systems for an experiment involves a lot of operational costs, and is not always relevant because of its small scale.

In the Chapter 5 we propose an I/O aware infrastructure simulation approach to explore the trade-off between the PFS and the DFS depending on the workload and platform.

### 2.5.3 Network

In the Big Data architecture, the network infrastructure is straightforward: Nodes are split in racks, with one rack per switch that is connected to the nodes with standard Ethernet links, and a fabric of multiple switches that interconnect the racks generally with a larger bandwidth.

There are a few references in the literature about the Big Data network. Nonetheless, a study about network requirement for Big Data [S A16] state that in contrast with client/server communication scheme that generates top-down traffic, data-intensive workload generates a lot of transversal traffic which requires a large uniform node-to-node bandwidth to efficiently communicates.

The HPC network often called the interconnect, is much more studied. The HPC interconnect is a whole class of very low latency, very high bandwidth network technologies that are mandatory to get peak performance out of HPC platforms: HPC applications are parallel tasks that communicate a lot for synchronization and waiting for communication is waste of computation power, and therefore a waste of time.

One of the main challenges with the HPC network is to scale while keeping low latency and high bandwidth. It requires special topologies that minimize the number of hops between nodes while keeping the contention at its minimum. It represents a whole field of research that will not be discussed here.

Due to a lack of time, the network part of the infrastructure alone was not fully explored during this thesis.

### 2.5.4 Virtualization and Isolation

Big Data clusters can be deployed on different types of infrastructure:

- **Bare metal**: directly installed on physical machines.
- **Hypervisor virtualization**: on virtual machines (VM) on top of hypervisors.
- **Container virtualization**: on containers on top of container aware Operating System.

Using bare metal is simple and provides performance efficiency because of the minimal number of software layer. Indeed, disk and Network Inputs and Outputs (I/O) latencies are impacted by the virtualization technologies [Fel+15].

But, with the emergence of the Cloud providers, the Infrastructure as a Service (IaaS) brings a level of service provides an on-demand cluster of VMs. Big Data clusters can be deployed on those virtual infrastructures and leverage the flexibility, management simplicity and low cost of this kind of service.

Recently the container technologies have arisen and have been popularized by Docker. The use of this technology has a much lighter virtualization overhead [Fel+15]. For Big Data workload, containers are preferred over VM because of there lower overhead on application performance [Bhi+17].

For performance reason, most of the HPC platforms are not using virtualization techniques and execute application directly on bare metal. But on-premise HPC platforms are very costly and there is more and more cloud providers that proposed an HPC offer (Amazon Web Services, Google Cloud, Microsoft Azure, IBM Spectrum Computing, Penguin Computing, . . . ). They are providing on-demand access to bare metal machines with high-end hardware and HPC interconnect.

A recent NASA report [NAS18] (2018) state that the cost of running in their own HPC center is from 2x to 5x less than running their workload on HPC clouds. Also, most of the applications they have tested run slower on HPC cloud due to less effective hardware, misconfiguration, or difference in libraries. In the end, HPC cloud is said to be useful for AI workloads that require specific hardware like GPUs, and to offload part of the home HPC center workload when there are bursts.

Even if the root component of the container technology, the Linux Control Groups are used in HPC RJMS since a decade for resource isolation and job cleaning, the container is only recently attracting HPC users for another use case: the file system isolation. It permits the users to build their own software stack (except for the operating system Kernel which is shared with the host) on a machine where he or she has not the permission to install anything otherwise. The network and file system isolation still has an overhead; it can impact memory bound MPI applications but the performance is getting better while Linux Kernel evolves [RJN15]. Thus, the container technology adopted on the HPC community recently but with a custom runtime that avoids the performances and security issues [JC15; KSB17b; PR17b; Gom+17a].

A more detailed study of the software stack provisioning on HPC in the context of the convergence with Big Data, using and virtualization technologies but also package managers can be found in Chapter 3.

# 3 Software Stack Provisioning for HPC Platforms

## Chapter's Abstract

The context of HPC and Big Data convergence creates a need for a systematic way to deploy a user-level software stack that can manage HPC, Big Data and hybrid applications. This chapter is a comprehensive survey on the software provisioning techniques applicable to HPC systems, with a focus on provisioning the tools data-intensive workload require. The chapter contains a definition of the different stages of software provisioning as well as the associated requirements. It then reviews the main approaches with respect to these requirements.

We conclude that while container-based solutions are a step forward, they failed to fit all the users' needs. In particular, we note the unwanted isolation that hinders access to hardware and reduce portability. The use of functional package managers appears to be a better match to the requirements of a hybrid software stack, with a more integrated, transparent, and flexible packaging methodology. Table 3.1 provides a summary of the different approaches with respect to the requirements outlined in this chapter.

## Chapter's Contents

## 3.1 Introduction

A software stack can be defined as a set of software layers required to run an application. Among them, we denote the application runtime, the application itself, and some libraries that the application relies upon (see the Section 2.3 for more details).

A major concern for users is to bring their software stack onto the cluster's machines, i.e. make their application run on the cluster. This process can be tedious because it may imply a lot of efforts from both users and administrators alike, in order to build, install, and run the entire software stack. This is the process that we refer to as software stack provisioning.

HPC and Big Data clusters have a lot of users, and each of them may have different needs - that require different tools. Both HPC and Big Data platforms are providing a set of pre-installed software packages that can be accessed directly by users. These tools are typically not sufficient for converged applications. This is due to differences between the HPC and Big Data software stacks: HPC software stacks are based on large parallel application, whereas Big Data stacks are based on a set of integrated services. The latter can be explained by the diversity of the tools that are needed for data processing pipeline: ingestion, storage, analytics, and archiving. Also, the resilience and dynamicity Big Data tools offer relies on third-party services like coordination and service discovery - which also need to be included in the software stack.

The provisioning process is also platform-dependent. An HPC platform is not only supercomputer hardware but also a set of services that are provided by the platform operators to the users. These services include the provisioning tool that is integrated with the other platform services. On a converged platform, where applications are not just HPC but also Big data applications, the provisioning process needs to be adapted accordingly. We need to find a provisioning tool for HPC platforms that is capable to fit the requirements of both data-centric and computation-centric application.

This chapter is a survey of the different software provisioning approaches that are currently used on HPC clusters. It also includes a reflexion on what is the most suited approach for a converged HPC/Big Data platform.

## 3.2 HPC: user requirements and constraints

### 3.2.1 Who's who on HPC platforms

To express the HPC users requirements, we need to define who the HPC users are.

- **Ordinary users:** Scientists from application fields (Physic, Biology, Climate, . . . ) use supercomputers to run large simulations that require the power of supercomputers. They principally interact with the platforms through a high-level interfaces such as web portals. They are working closely with research engineers on scientific applications. Their main goal is to extract reliable scientific results from HPC platforms.

- **Power users:** Scientists who are experienced enough to use a lower level interface to interact directly with HPC. They require a higher level of customization and are willing to gain more knowledge about HPC systems. In the same category, we can find Research Engineers that develop and maintain scientific applications. They share the same goal with regular scientific users, yet they are concerned with the intermediate goal of making the system work in a resource efficient way.

- **Computer Scientists:** Computer scientists that work on HPC applications. Their needs are similar to those of the Power Users, but their goal is different. They are studying HPC applications as a research topic, trying to improve their efficiency on existing or design future HPC platforms.

- **Operators:** The operators are in charge of the HPC platform. They maintain it and develop it according to users' needs. In close collaboration with the HPC vendor, operators are proposing a set of services to the end users, and answer to their potential requests. Cluster operators set the job scheduling policies, and develop and maintain the software stack that is installed on nodes.

- **HPC infrastructure Computer Scientist:** This specific class of users work on the HPC system itself as an object of research. To do so, they need a high degree of control, and depending on their topic of research they may have requirements that are similar to those of the operators. As production HPC platforms are expensive and constrained, they are not working on an entire platform but on dedicated testbeds, where they can reach the level of control required to run faithful experiments (detailed in Section 3.6.1).

In the following, the term "HPC user" will refer to any of the first 3 users group without distinction.

The following subsections dissect the software provisioning process according to the different steps of which it is made up: build, install, configure, run, package and share. For each of the provisioning steps, the subsection includes the user's and operator's wanted features for hybrid Big Data/HPC applications. Each feature described in this section has an attributed code (e.g. B5) that is reported on the summary table in the conclusion of this Chapter 3.1.

### 3.2.2 Build

The first step of the provisioning process is for the user to compile, or build, his software stack. But, as HPC users are scientists, scientific software that are built for HPC should produce reliable and reproducible results. In computer science we ensure reproducibility with a deterministic software building process: a fixed set of parameters in input should produce a bitwise reproducible binary (B3). This eliminates one of the sources of variability. Another main concern for reproducibility is the traceability of the build process. Someone who wants to rebuild the application needs to know how the application build process was done and what are the build dependencies (B2). These dependencies are defined as package names and need to be resolved to actual packages locations on the build machine (B6). This resolution depends on the package version of each dependency. This is only one parameter of the build process. HPC users need to deal with many parameters, among which:

- Platform
  - Architecture (x86_64, Power8, ARM64. . . )
  - Interconnect (Infiniband, OmniPath, BXI, . . . )
  - Accelerator (Nvidia GPU, AMD GPU, Intel KNL, . . . )
- Software dependencies
  - Compilers (GCC-6, GCC-7, Clang, Intel, . . . )
  - MPI libraries (OpenMPI-1, OpenMPI-2, mvapich, IntelMPI, . . . )
  - Other libraries (OpenBLAS, OpenMP, . . . )
- Compilation option (for each software piece)
  - Configuration flags (–enable-foo, –enable-bar, . . . )

  - Compilation flags (-O2, -g, . . . )

Even though some are often fixed, HPC users are facing a combinatory explosion of the build process parameters. We need a tool that provides an easy way to explore build parameters (B4).

Part of these parameters are imposed by the HPC platform: the architecture, the OS, and sometimes the version of the MPI library that works with the local resource and job management system (RJMS), and the local interconnect. Some applications or libraries are made to last longer than the HPC platforms however, and as a consequence these software projects need to be portable (B1). Platform constraints are then seen as build options.

In order to optimize the selection of those parameters for a certain platform, some of them can be guessed when the build process is done on the targeted platform (B5). Provisioning systems need to be able to make this choice and build on the platform without special privilege (B7).

### 3.2.3 Install

The installation process involves both operators and users. In fact, both parties need to trigger installations on the system but not with the same objectives.

In order to avoid storage and memory waste, operators want users to share libraries of their software stack if possible (I7). They also want to push security updates and new versions of tools to the users' software environment (I9), but this implies that the users are using operator defined dependencies (I2).

In the other hand, users want flexibility: They want to have the permission to install whatever they want (I1), from source (I4) or binaries (I3) and possibly to have multiple versions of the same software (I5). They also may want to have multiple environments to install different sets of packages (I8) with different configurations for different use cases (development, tests, production, . . . ). HPC users also want to install and remove packages without breaking anything: They want a transactional environment modification model with a rollback capability (I6).

### 3.2.4 Configure

Once software is installed, it needs to be configured. The configuration phase can be minimal or nonexistent for traditional HPC software, where the application's configuration parameters are set at build time. However, most of the standard Big Data software are compiled to run on the Java Virtual Machine (JVM). Using the JVM abstraction simplifies the provisioning process by avoiding the complexity of the compilation phase, because the only target is the JVM. Both the JVM and the application then need to be adapted to the local platform architecture. Thus, most of the compilation complexity shifts from the build phase to the configuration phase. Moreover, Big Data tools may have lots of configuration parameters. For example, Hadoop -the Big Data framework which contains Yarn, MapReduce, and HDFS- has more than a thousand configuration properties listed in its documentation (2.7 version). Moreover, data-centric computation is based on a data pipeline that includes multiple elements that need to be tightly integrated together. This integration is also a matter of configuration that needs to be taken into account by provisioning tools.

A provisioning tool should permit to explore all those parameters (C3). Moreover, the configuration used at run time needs to be traceable in order to be able to reproduce the results of an experiment (C2). The provisioning tool should also ensure that configuration files are portable if possible, i.e. that it does not contain a hard-coded path pointing to one user's home environment (C1).

Platform customization is important and platform operators should be able to provide a sound configuration adapted to the platform that the users can easily override as needed (C4).

### 3.2.5 Run

The run phase is the final goal of the provisioning. It depends on the 3 previous phases: Build, Install, and Configure. Still, there are some issues that need to be taken into account.

First, the users want to be sure that the shared libraries used at build time are also used at run time (R1). Indeed, because the build environment can be different from the run environment, any shared libraries that is loaded through the linker at launch time may be different from the ones used during the build. This may lead to unwanted performance drift, changes in the application behavior, or even application crash.

Secondly, traceability is also wanted at this stage, in order to have complete information about the run. The software stack that is used including the shared libraries, the configuration files, the inputs and the outputs of the application all need to be traceable(R3).

Finally, the isolation provided by the Linux namespace hinders the user's application processes capability to access to the hardware and file system, and should thus be avoided (R2). This is explained in more detail in the container section (3.5). Also, when an application is launched through a container runtime, a particular command line interface with multiple options needs to be used to be able to circumvent the aforementioned limitations. A provisioning tool should not modify the user experience (UX) because it breaks compatibility with launching tools and increase the users' cognitive load (R5).

### 3.2.6 Package and Share

Packaging and sharing are part of the provisioning because it is the ability to bundle the user software stack in some kind of package and move or share it with other users. In order to do so, HPC users need to be able to export a binary version of their entire software stack in a self-contain package that also includes configurations (P4). Moreover, it is necessary to be able to export any package set into a container image in order to ensure portability to Cloud-based platform (P6). Finally, for HPC operators and users, a large Internet distribution, through private (P7) and public (P8) repositories greatly simplifies software stack sharing.

This ability to move a package from one place to another can also be used by the user that made the package in the first place, in order to move his software stack to another platform. Because most of the HPC software needs to be recompiled with different parameters when it lies on a different platform, this requires the capacity to re-build from source (P3) with modifications (P5).

For reproducibility issues, the entire package creation process should be traceable (P1), and strongly reproducible even on another platform (P2).

### 3.2.7 Summary

We have defined all the provisioning features that convergent HPC platform users and operators would want. Some of them, especially the ones from the configuration stage are related to new data-oriented use of the HPC platform. The rest of this chapter will be looking for provisioning tools that match the entire feature set - including both traditional and data-centric usage.

Several software stack provisioning tools have emerged in the HPC community. The following sections describe the different approaches that try to solve these issues and how well they tend with the requirements described above.

## 3.3 Traditional Package Managers

As of November 2017, all HPC machines from the Top500 are running a Linux based OS, and most of them are shipped with a RedHat based distribution [Top18]. This distribution, like most of the others, is based on a package manager that makes heavy use of shared libraries (libraries that are shared among all the users' environment).

The fact that libraries are shared prevents to use this kind of tool for a user-defined software stack, because it imposes the use of one particular version of each library. Moreover, it is not possible for a regular user to use these package manager because it would alter the state of the software stack for all users, which may lead to silently break other user's application.

From the HPC operators point of view however, package managers are well suited for system management: They bring guarantees from the distribution provider that all packages are working together. Thus, they can provide a stable and validated software environment basis for users. Additionally, when a security flaw is discovered, the package distributor may provide security patches over the stable version of the package, a perk which is not always provided by upstream package developers. These updates may allow to avoid breaking users' code, by only applying minor security fixes that do not impact the binary compatibility of the shared libraries. For all these reasons, package managers are a good choice for cluster operators.

Still, HPC users need to work on their own specific software, but without the privileges to change the libraries and programs available on the compute nodes - traditional package managers cannot be used for user software provisioning.

## 3.4 Virtual Machines

Virtual machine (VM) brings heavyweight isolation: The VM operating system is seeing the machine interface provided by a Hypervisor exactly as if it was a hardware interface. This kind of isolation improves security by keeping users on a separated OS, and flexibility because VMs can migrate from node to node and their state can be checkpointed and restored. However, they also come with overhead on resource accesses which have to go through virtualization layers. This aspect has been improving: hypervisors like Qemu/KVM, in conjunction with the hardware support for virtualization, greatly increase the performance of VMs over the last years. One of the key mechanism to avoid virtualization overhead is *para-virtualization*: This means that the guest OS is aware that it is running on virtualized hardware and thus avoids duplicating work with the host OS. The *virtio* and *vhost* para-virtualization modules provide better performance than unaware virtualization for disk and network IO throughput. Felter et al. [Fel+15] show that memory access in a VM suffers from higher latency than to native access due to memory page lookup on the guest and on the host. Also, computation overhead can go up to 17% for the Linpack benchmark in simple configuration setup, but this is mitigated by CPU 1 to 1 pinning that allows reaching almost native Flop/s (-2%) for Linpack. Still, particular use cases like compression with the PXZ benchmark suffer from an 18% overhead. The main remaining problems in VM performances are network and disk IO latencies that lead to more than 2 times lower performance for random read operations.

In HPC centers, VM are sometimes used for security purposes in production as well as testing and prototyping - all of which leverage the flexibility of virtualization. Hovwever, machine virtualization use is often avoided for computational tasks because of the performance issues detailed above. As a closing note, IO latency performance issues make VM technology a bad candidate to host Big Data applications.

## 3.5 Containers

Container technologies were implemented by many operating systems since the 1990s with Linux chroot, FreeBSD jail and Solaris Zones. They have been used for lightweight resource sharing on shared hosting machines. These technologies were recently made popular by the advent of cloud computing. The Linux container implementation was designed and pushed in the Linux Kernel by Google and Canonical, and popularized by Docker which made it easy to use thanks to the concept of container images. Container images are made to be a generic packaging solution for services. It is a commonly used abstraction for service deployment and orchestration.

From a technical point of view, a container can be seen as lightweight virtualization at the OS process level. In fact, it is a process isolation technology based on the OS kernel features. It means that containers that are running on the same machine are sharing the kernel like any other process, yet they are placed in a resource container that only expose part of the hardware resources (e.g. 1 CPU core instead of 16, 4GB of memory instead of 128GB) and gives an virtually isolated vision of the system (e.g. a dedicated software stack, an isolated list of process, altered mount point, . . . ).

Containers have less overhead than VMs, and their performances is natives especially if some isolations, like the network isolation, are dropped [Fel+15]. Indeed, the main problem with containers in HPC is not performance but isolation. Provided by the Linux Kernel Namespaces, this isolation was originally made for web services that only interact with the outside world through network sockets, not for high-performance parallel applications. The list bellow, gives details about the different isolations implemented by the container runtimes and the problems caused by each of them for HPC applications:

- *Mount*: This is the most wanted isolation because it permits to isolate the entire software stack from the host system. But, it causes a lot of trouble for special hardware support or any piece of software that need kernel drivers with the associated libraries that depends on the kernel. Thus, some part of the host file system must be exposed inside the container to be able to support for example the GPU or the interconnect's network interface. It also prevents to load and share files from the user's home space which can be painful for users. A bind mount of the users home inside the container may solve this issue but creates permission problems because of user id isolation.

- *User ID*: Users are root inside the container to be able to install and run whatever they want but all the files exported from the container might have wrong UID/GID and create permission problems. This namespace has an unstable feature that permits to map the external user and group's ID from the outside to another user inside the container in order to fix permission issues. But, due to security issues (CVE-2014-4014, CVE-2014-5207, CVE-2015-4177, CVE-2017-1000111) major Linux distributions disable this feature in the Kernel by default.

- *Process ID*: This is the isolation of the process id: It means that from inside the container, the only processes that are seen are the ones that run inside the container. One of the possible problems is that it prevents process cooperation. For example, an MPI coordination process, which is widely used in HPC, cannot run from inside a container because it would not be able to find other local processes.

- *Interprocess Communication (ipc)*: This isolation, like process ID isolation, can harm process collaboration because it is needed to share memory between processes.

- *Net*: The network isolation is using virtual interfaces and bridges to connect the containers with the host and/or the rest of the nodes. This level of virtualization leads to poor network performance [Fel+15].

Another problem is the fact that creating Linux namespaces requires super-user permissions. It means that a normal user has to use a tool that has root privileges in order to launch a new container, which is a possible security leak. In recent Linux Kernels (Since Linux 3.8), User namespaces give the possibility to create all the aforementioned namespaces from a non-privileged process that owns the `CAP_SYS_ADMIN` kernel capability. However, the Red Hat Enterprise Linux distribution - which is widely adopted in the HPC community [Top18] - has disabled user namespaces for non-privileged users for security issues in the current release (7.x).

To circumvent these issues, HPC centers have developed their own Container engines that minimize the number of namespaces used or even avoid them completely, and cope with the possible security issues. Here is a list of these tools.

### 3.5.1 Singularity

Singularity is a container solution initially developed at Lawrence Berkeley National Laboratory (US) for their own needs. It is now backed by a company, and it is becoming a bigger project that includes an image generator and image repository. Singularity is currently installed on many HPC sites [KSB17a].

A user of Singularity needs to create a container image on his own machine because privileges are required by the image creation process. Then, the Singularity image can be run without privileges, like a simple process on the HPC cluster. However, the Singularity launcher is using the SetUID instruction that requires the root privilege at installation time: It means the HPC operator needs to install Singularity on each node to permit the users to run a Singularity container. Singularity also comes with user namespace support, but as mentionned earlier this namespace is not supported at HPC sites.

Each container is a simple process isolated by a mount namespace created by a wrapper around the process, or by the singularity CLI. With this mechanism, Singularity is removing the privileged daemon that Docker, one of the industry leaders, uses to manage containers.

Still, Singularity is not able to simply circumvent the unwanted isolation issues because some libraries present on the host node have to be imported inside the container in order to work properly. The most prominent example is the MPI library, which is ubiquitous in HPC applications: an MPI application running with Singularity is called from outside the container through the `mpirun` CLI. Thus, the main MPI process (ORTED) that coordinates all the distributed process across the different nodes is not in the container and must be ABI-compatible with the MPI implementation that is inside the container.

### 3.5.2 Charlie cloud

Charlie Cloud is an approach from the Los Alamos National Laboratory (US) that tries to provide the simplest tool based on the HPC users and administrators requirements they defined [PR17a]. They choose to use only the User namespace that is declared mature enough, to enable the change of root without privileges. It requires Docker to manage the building and sharing of the container images and implies that this process is done on an environment where the user has privilege, like his own machine.

An interesting feature is the capability to run an MPI application with the process coordinated from outside the container, or from inside the container to avoid compatibility issues. But, it has the same problem as Singularity for the outside coordinated version. For inside coordination, it exposes the application to potential performance loss due to MPI misconfiguration regarding the platform. Indeed, most of the HPC sites are providing their own version of MPI that is properly integrated with the RJMS and the interconnect.

The UNIX philosophy approach of this tool makes it appealing. However, as mention above, the user namespaces are not available on most HPC sites due to security issues, which makes this solution unusable there.

### 3.5.3 Udocker

European HPC centers also develop a container solution for HPC: udocker [Gom+17b]. The idea behind this approach is to design a tool that requires no special privilege or capability to run a container: it permits users to run a container without involving administrators. To do so, udocker offer different approaches: Use the user namespace (if available), or avoid the change of root provided by the namespace and use a mechanism to intercept system calls that handle pathnames. It means that the image is unpacked on the user's home space and run without system isolation. Thus, absolute paths that are present in the container image binaries need to be relocated to the current location without recompilation. There is two possible mechanisms to achieve this:

1. Using the Linux kernel interface called PTRACE thought the PRoot tool that was modified to use the SECCOMP filtering capabilities of Linux after the 3.8

version. This optimization makes the system call interception more precise by filtering only the ones where pathnames are involved.

2. Using the `LD_PRELOAD` variable to load a special library called Fakechroot that was also modified to handle all the case that cover udocker. It also automatically modifies the binaries present in the container using PatchELF utility to prevent accidental loading of the host's system libraries.

According to the udocker paper [Gom+17b], only the first approach has a performance penalty, which is comparable to that of Docker. The impact can be higher especially for I/O bound applications because performance loss is proportional to the number of I/O system call. This first method is simpler and interoperable with older kernels but it has a performance cost. The second approach has no significant performance loss and provides the same level of isolation as the functional package managers described below, in Section 3.7.

Even if the user namespace is not used, this solution has the same portability issue as the others because the MPI version inside the image must be the same as outside the image, in order to run properly.

### 3.5.4 Container integration to RJMS

This is a comparison between the different type of containers integration within a Resources and Job Management System (RJMS). The following is a list of the existing approaches.

#### Shifter

Shifter is a tool presented by the National Energy Research Scientific Computing Center (NERSC) in 2015 [JC15]. Shifter is integrated with the open source RJMS, SLURM. It allows Cray HPC users to specify the Docker image they want to use through a custom argument in the submission command line of SLURM. The specified image is imported into a local image management tool. The Lustre file system is used to deploy the user-defined image that is customized to achieve local configuration requirements. Then, the image is mounted on a loop device in order to be immutable, and a prologue script activates the image with a *chroot*. This mechanism provides the file system isolation while the resource isolation is left to the RJMS which already implements it for normal jobs.

This system appears to be tightly integrated with the NERSC Cray infrastructure. This level of integration allows the container images to be used almost transparently by HPC users because the container launch is made by RJMS and not the users.

### Socker

Developed at the University Center for Information Technology (USIT) in Norway, Socker [Aza17] take another approach: Instead of replacing the Docker daemon with a simpler implementation tuned for HPC needs, they choose to integrate it with the RJMS. In fact, most of the RJMS, including Slurm, are already featuring an isolation of resources mechanism using the Linux CGroups, and the idea of the author of Socker is to run the user's containers in these CGroups. To do so, Socker contains an executable launched by the end-user alongside with the resources request to Slurm. This executable collects information on the user (UID/GID) and on the container to run (image name, command), then it calls SetUID to start the docker container inside the user's job with the user's IDs. It means that a docker daemon is running on each node but only the administrator of the cluster and the Socker executable are able to use it directly. The paper also provides a performance evaluation of this solution and shows that its performance overhead is the same as using Docker directly.

This minimalist approach fits in a less than 300 lines of python (which is compiled to avoid security issues), and reuse a maximum of existing tools and mechanism. Socker is a proof of concept still under development with a small community. In comparison to Shifter, the use of Docker prevents image customization and adaptation to the platform.

### Singularity

Singularity also provides an integration with Slurm through a plugin. Like Shifter, it permits the users to run a job inside a container with the image of his choice, but it uses Singularity as container runtime.

### 3.5.5 Summary

Containers technologies are seen as a point of convergence between Big Data and HPC infrastructure [Asc+18]. Indeed, it is already the standard way to provision service-oriented infrastructures in the industry, and it a natural candidate for the deployment of Big Data stack on HPC.

But, if the containers' isolation is a wanted feature in Cloud infrastructure, it leads to a lot of issues for HPC applications. Some HPC oriented container runtimes are trying to remove this isolation entirely, but the mechanism used to circumvent the Mount namespace may imply a loss of performance, and add complexity at execution time, i.e. it is not transparent for users.

Solutions integrated with the resource manager, e.g. Shifter, seems to be more suited for HPC usage. Because it manages the container launch, this kind of system can integrate the locale platform constraints in the container image automatically, thus HPC users don't have to manage platform specific options by themselves.

Finally, for converged Big Data/HPC applications we can assume that HPC technologies are used underneath. The problem is that HPC applications need to be adapted to each HPC platform in order to be able to run efficiently. This adaptation generally requires recompiling the application. Containers are not package managers, and they do not manage application build at all. It means that container images may be a convenient package format, but it needs to be used in conjunction with build tools to achieve portability. This statement greatly diminishes the container's usefulness on HPC platform.

## 3.6 HPC provisioning

The HPC community has built several tools for software provisioning that are well integrated into the HPC ecosystem. They are made to fit HPC needs that emerge over time.

### 3.6.1 Bare metal provisioning

In Computer Science research, scientists need to fully manage the software stack: from the Firmware to the application code including the operating system (OS). Emulab [Whi+02] and Grid5000 [Bal+13] are testbeds for computer science that are designed with this requirement in mind. They provide the capability to deploy an OS image directly on *bare metal* (e.i. On the machine hardware). With the ability for the testbeds' users to reserve machines, the testbed also enables reservation of networks (VLAN, subnets) thought switch instrumentation, of storage space on a network file system (NFS), and even of disks on some machines.

In order to achieve experiment reproducibility, Grid5000 provides a different flavor of OS images that can be deployed on the user's allocated nodes. Those images are built in a reproducible manner using Kameleon [Rui+15], and can be extended by the users to add their specific software stack.

**Chameleon** [MCY15] is another testbed that provides an Infrastructure as a Service (IaaS) interface based on OpenStack [Opeb]. It proposes bare metal deployment of OS images created by an OpenStack tool called diskimage-builder [Opea]. Additionally, Chameleon provides software-defined network (SDN) to their users, to give them a close control over the network.

With this type of provisioning, a Big Data stack can be installed and configured directly on the OS images. However, some configuration must be performed after the OS deployment. This can be done manually, which harms reproducibility, or using scripts. The Grid5000 team is supporting tools that are made for this end [Gri18].

These testbeds are fulfilling the specific needs of complete reproducibility and control that computer science experiment requires. They are not representative of production HPC platforms, which do not provide this high level of customization. In fact, HPC administrators prefer to keep control of the OS deployed on the nodes to minimize both security issues and maintenance cost. For this reason, bare metal provisioning is restricted to cluster bootstrapping (i.e. first installation) and Computer Science experiment testbeds for HPC scientist as defined above 3.2.

### 3.6.2 Do It Yourself

This not a tool but a simple method. The most obvious way for HPC users that are developing their own software is simply to compile it inside their private home space on the HPC platform's login node. They can choose to use the libraries provided by the existing environment or use their own custom libraries they installed on their home space. It permits the user to mix defined software with platform provided tools easily.

This approach is straight forward and gives great flexibility to users. On the other hand, end-users have to manually build their own software stack, which requires technical expertise. This build process is also not reproducible: It is based on a series of custom commands that depends on both the HPC provided environment and the user's external tools imported on the cluster.

Because this method is not standardized, with no traceability of any stage of the provisioning, it is to be avoided when possible.

### 3.6.3 Environment Modules

The aforementioned process that lets HPC users build their own tools manually could be compromised by the very first step: bring dependencies to the user environment. Indeed, the number of dependencies for production application may be high and compiling every dependency may take hours. To cope with this major problem, HPC systems administrator use **Environment Modules** [FO96]. This tool provides a command line interface (CLI) that give to the HPC users the capability to load or activate a set of modules. These modules are defined by a TCL script (or Lua with the LMOD implementation [McL+11]) that modifies the user environment to make an application or a library available. It is compatible with most shells (bash, zsh, fish, ...) and some interpreted languages (Perl, Python, Ruby, ...) through initialization scripts.

One of the major caveats of Modules is that activation scripts are changing the current environment with side effects. It means that loading and then unloading a module does not warrant that the user's environment is completely back to its initial state: The outcome depends on the module's implementation. Also, the user environment at compile time and run time depends on the local cluster's sets of modules. Thus, the experiment environment's construction and the experiment itself are not reproducible and the fact that the exact same modules should be loaded at compilation time and at run time is error prone. Furthermore, even if *Environment Modules* solve the problem of dependencies, the issue of building the user's software remains.

Some tools have emerged in the HPC community to allow trying all the relevant combinations in a reproducible manner.

### 3.6.4 EasyBuild

**EasyBuild** [GHM14] is such a tool. It only manages software builds, and it hinges on *Environment Modules* for software activation. Factually, EasyBuild is a tool that simplifies the creation of environments through Python modules. Those Python modules are defining a set of parameters for the build. Those parameters, plus metadata, are given as configuration files that are manipulated directly by the end-users through the EasyBuild CLI. Each set of parameters that the user wants to build is defined by one of those configuration files. It is also possible for some parameters to avoid the configuration file by providing parameter thought CLI.

With EasyBuild, the build phase of the provisioning is managed, but a lot of features are still missing. Also, it does not provide any guarantees regarding the reproducibility of the build and the run phases.

### 3.6.5 Spack

The Supercomputer Package manager named **Spack**[Gam+15] is similar to EasyBuild in the sense that it is based on Python modules that define the build process. These modules can be seen as templates that provide a different way to build software depending on the build parameters. Spack does not use configuration file but a powerful CLI options grammar which allows the user to select the build parameters of the tool itself but also of all its dependencies. Thus, Spack focuses on managing the combinational explosion of these parameters.

Spack also supports an arbitrary number of application and library installations, even with the same version number, using cryptographic hashes to identify each installation. This identification system permits Spack to support re-utilization of previous builds for common shared libraries.

Another issue that Spack is solving is the problem of the difference between the building environment and the execution environment. To do so, it explicitly links the libraries that are used at build time inside the produced binaries, using the RPATH header section of the ELF binary format. This approach brings together the advantages of static and shared libraries, sharing really identical libraries, and supporting multiple version of the same library even if the difference is just a compilation flag.

To resolve package dependencies that can be represented by a directed acyclic graph (DAG), Spack uses user's inputs in the Python module that defines the package. The dependencies resolving is done in a special phase, called *Concretization*, which depends on the package definition and the options given through CLI.

For the installation phase Spack generally relies on *Environment Modules* 3.6.3.

### 3.6.6 Summary

Most advanced HPC provisioning tools are made to customize application build and easily install it locally. Thanks to recipes that are implemented in a standard programing language, each package can be customized by users to fit their needs.

Spack is the most advanced tools for build parameters management with its innovative CLI and also provides a good build reproducibility guarantee with portability capabilities. For the installation phase, even if it relies on Environment Module that does not give run time guarantees, binary patching permits environment to be loaded and unloaded safely.

However, with the new data-intensive usage, applications become more complex and the configuration phase need to be taken into account. Those tools are highly specialized for HPC needs and are not capable to manage configuration parameters.

## 3.7 Functional Package Managers

Another recent approach for provisioning application on HPC is the use of functional package managers.

### 3.7.1 Nix

Some concepts of Spack, like hashed installation folders and RPATH editing, were previously proposed by the purely functional package manager called **Nix**[DDV+04]. Nix is defined as purely functional because a package is defined by a deterministic function written in the Nix Expression Domain Specific Language (DSL). Nix uses the concept of pure function – a function that is deterministic and does not trigger side effects – and applies it to the software provisioning problem. The absence of side effect means that all changes made on the file system are constrained to a special location called the Nix store. Each build of a package, produce a set of files called derivation, that is stored with a hash of all the build inputs as a prefix. Packages are then installed just by creating symbolic links in the users home or by changing environment variables.

Dependencies are resolved using the function parameters only. Thus, Nix avoids costly dependency solving algorithms. However, all the packages inputs need to be defined as Nix packages, which forces the users to declare all the dependencies in Nix. It has the drawback that if a dependency package is not described in Nix expression yet, the user has to write this package in order to able to build his own. Therefore, it provides the great advantage to force the package itself, and its dependencies to be defined from source, which is a good property for reproducibility. Another advantage is that the complete software stack of an application is easy to extract as a closure. A closure is a simple tarball that can be imported in any other machine that has Nix installed. Furthermore, it enables complete isolation during the build process (using the `build-use-sandbox` option), which enforces the fact that the package closure is truly self-contained.

As it is a general purpose package manager, Nix was not designed for HPC. Thus, it does not provide a Spack-like convenient CLI to manage build parameter complexity. However, the parameter exploration capability of Spack can be reproduced declaratively using the overriding mechanism embedded in the Nix package functions: One can override package function inputs to change build dependencies, or even override the package attributes inside the function, e.g. change the source code location, or the configuration flags. This creates a new separate package definition without erasing the previous one.

Nix also propose the notion of environment profiles with transactional updates and rollback. Each user can have several activable profiles where installations can be made without privilege. This mechanism is heavily based on symlinks that make software from the Nix Store available by creating pointers to the wanted tools in the user environment. The profile atomicity and rollback features are the consequence of the side-effect-free nature of this activation process. The strength of this approach for creating software environments lies in its simplicity: It does not come with unwanted isolation that hinders user's capability to interact with the file system or the machine's hardware (see Section 3.5).

Configuration can also be taken into account by Nix packages using a wrapper mechanism. One can create an application wrapper package that takes this application with a configuration in input and create an executable script that launches the application with the applied configuration. As a wrapper is a package, any change in the configuration is declarative and will create another distinct package without rebuilding the original application package. This mechanism allows to simply creates a fully configured application, or even integrated set of applications that are traceable and reproducible. This last feature is critical for data-centric application that needs configuration and integration to be packaged properly.

Moreover, Nix eases the experimental workflow because it brings full reconstructability of the HPC experiment software environment: A scientist can build the entire software stack of his experiment on his local machine, then export the entire closure if this stack (the final software and all its dependencies with their configurations), push this closure to the HPC cluster, use Nix to activate the environment, and finally run the experiment.

### 3.7.2 Guix

In 2013, a new approach based on Nix called **Guix** [Cou13] was proposed. It supports the same features as Nix and it uses the Nix build daemon under the hood.

The main difference with Nix is the language used to define the packages. In Nix, a simple DSL called Nix expression was created for this purpose. The whole Nix packages are written in this language but also heavily relies on bash scripts under the hood. Guix uses an embedded DSL, i.e. backed by an existing programing language: Guile [Gui19]. This permits to use the language tools and ecosystem, instead of building one, like for Nix. Guix also completely replaces bash scripts by Guile implementations, which improve error handling and reliability. However, as the Guile language adoption is narrow, the advantage of this choice is not conclusive. Guix also provides a well designed CLI that outperform the Nix CLI in terms of user experience.

A recent effort called GuixHPC [Cou18] is proposing to use Guix for HPC reproducible software deployment.

### 3.7.3 Summary

Functional Package Managers provide lots of original features like userland transactional installation, the possibility to creates shared (or private) virtual environments for each project gives the same level of functionality as the containers, but at a finer grain and without the hassle of image management that comes with containers.

They mostly provide the same features as HPC provisioning tools but some differences are worth noticing: Functional package managers are not capable to use software that is not defined as package function, thus platform specific libraries also need to be packaged - which is a supplementary task for HPC operators. On the other hand, by forcing all dependencies to be packaged, they provide strong reproducibility guaranties and simplifies the software stack extraction. Functional package managers are not able to cope with an applications' configuration, which is not the case for HPC package manager. Finally, operators are unable to push security updates when the platform packages are not defined in Nix, and only Guix, for now, is capable to do this without recompiling every software which depends on the library that was fixed.

Functional package manager do come with the cost of learning an associated language which is not widespread in the HPC community. Indeed, most advanced features force the users to write code, which can be a major problem for newcomers.

Nix adoption in HPC is narrow but growing recently in computing centers [Bze+17] and in larger installations like the Blue Brain Project [DDS15a]. With the Guix HPC initiative, Guix is now (2018) deployed on 4 European HPC clusters [CW15b; CW18].

## 3.8 Conclusion

The best approach for software provisioning of the Big Data software stack depends on two factors: The level of integration of the Big Data tools that the user wants to use in the HPC provided environment, and the complexity of the software stack. In fact the HPC users software stack is becoming larger as the science becomes more data-oriented: Managing more data both in inputs, with the growing number and quality of scientific apparatus, and in outputs, with the increasing accuracy of simulation models, leads to a change of scale in the data management. This change of scale implies to deploy and integrate a full data pipeline, from data ingestion to data analytics. This pipeline is composed of a set of independent component that needs to interact closely. In this trend, the prominent problem is not just compilation anymore, like in traditional HPC, but managing configuration and integration properly. In order to do so, we need a set of tools that package data-oriented applications and ties them together with the necessary configuration glue.

Software provisioning tools designed for HPC tools are not suited for this task, and so the HPC community is looking for the solutions that come from the world of the Cloud: VMs and containers. A great benefit of these technologies is to add a level of abstraction that enables users to easily use well-known package managers to build their own software stack in an isolated environment. Like every abstraction layer however, these techniques come with a cost on performance and operational complexity. Indeed, the isolation that comes with VM is not wanted on an HPC cluster, mostly for performance reasons. On the other hand, the main issue with containers is the fact that a container image built somewhere need to be rebuilt for every platform to fit with the special hardware drivers and the cluster supported MPI version, thus defeating its main purposes of portability and ease of use.

Package managers like Nix and Guix are hard to understand for newcomers because the disruptive way they handle package's definition, i.e. using a functional language. But, as shown in Table 3.1 the functional package management approach covers most of the needs for complex scientific software stack provisioning, including configuration management. Providing a more user-friendly interface to those tools - which would not requires the user to write code - should increase functional package manager adoption in the scientific community.

|  |  | **Containers** | **EasyBuild** | **Spack** | **Nix/Guix** |
|---|---|---|---|---|---|
| **Build** | *B1:* Portability (cross build) | (✓) using Qemu | ✓ | ✓ | ✓ |
|  | *B2:* Traceability (How it is built?) | (✓) need recipe | ✓ | ✓ | (✓) need the Nix expression |
|  | *B3:* Bitwise reproducibility | ✗ | ✗ | (✓) if no external module used | ✓ |
|  | *B4:* Parameters exploration | ✗ | (✓) | ✓ | (✓) |
|  | *B5:* customization by platform | ✗ | ✓ | ✓ | ✓ |
|  | *B6:* Dependencies management | ✗ | (✓) no guaranties | (✓) no guaranties | ✓ |
|  | *B7:* No privilege needed | ✗ | ✓ | ✓ | ✓ |
| **Install** | *I1:* No privilege needed | ✗ needed namespace setup | ✓ | ✓ | ✓ |
|  | *I2:* Mixed stack (from platform + from user) | (✓) using bind mounts | ✓ | ✓ | (✓) only if platform stack is defined in Nix |
|  | *I3:* Binary install | ✓ | ✗ | ✓ | ✓ |
|  | *I4:* Source install (includes local build) | ✗ | ✓ | ✓ | ✓ |
|  | *I5:* Multiple version of the same software | ✓ | ✓ | ✓ | ✓ |
|  | *I6:* Atomic operation | ✓ | ✓ | ✓ | ✓ |
|  | *I7:* Share user install with other users | ✗ | ✗ | ✗ | ✓ |
|  | *I8:* Virtual environment | ✓ | ✓ using module | ✓ using spack env | ✓ using nix profiles |
|  | *I9:* Security updates by operator | ✓ | (✓) only for external packages | (✓) only for external packages | (✓) only in Guix for now |
| **Configuration** | *C1:* Portability (Can be moved and shared) | ✓ | ✗ | ✗ | ✓ |
|  | *C2:* Traceability (What is the configuration used?) | ✓ | ✗ | ✗ | ✓ |
|  | *C3:* Parameters exploration | (✗) need entire rebuild | ✗ | ✗ | (✓) with Nix expression |
|  | *C4:* Customization by platform | ✗ | ✗ | ✗ | ✓ |
| **Run** | *R1:* Build time equals Run time guaranties | ✓ | ✗ | ✓ | ✓ |
|  | *R2:* No unwanted isolation | ✗ | ✓ | ✓ | ✓ |
|  | *R3:* Traceability (What is running exactly?) | ✗ | (✓) using module | (✓) with spack specs | (✓) with nix-store -q |
|  | *R4:* Native performance | ✓ | ✓ | ✓ | ✓ |
|  | *R5:* No UX modification | ✗ | ✓ | ✓ | ✓ |
| **Package and Share** | *P1:* Traceability (How the package is built?) | ✓ | ✓ | ✓ | ✓ |
|  | *P2:* Strong package reproducibility | ✗ | ✗ | ✗ | ✓ |
|  | *P3:* Source package sharing | ✗ | ✓ | ✓ | ✓ |
|  | *P4:* Binary package sharing | ✓ | ✗ | ✓ | ✓ |
|  | *P5:* Easy to modify package | ✓ recipe script | ✓ Python | ✓ Python | (✓) Nix expression |
|  | *P6:* Export as a container image | ✓ | (✓) experimental | ✗ | (✓) with Nix expression |
|  | *P7:* Private binary package repository | ✓ | ✗ | ✓ | ✓ |
|  | *P8:* Public binary package repository | ✓ | ✗ | ✗ | ✓ |

Table 3.1.: Summary of software stack provisioning process features and their implementation for the main provisioning tools. Each feature is described more precisely in Section 3.2

# BeBiDa: Best effort Big Data on HPC cluster

4

## Chapter's Abstract

Recently, several efforts are made to make HPC and Big Data coexist on the same platform using different approaches. This chapter first discusses these approaches and proposes a new one based on Resource and Job Management System's (RJMS) collaboration.

In classic HPC workloads, the rigidity of jobs tends to create holes in the schedule: we can use those idle resources as a dynamic pool for the Big Data workloads. We propose a new idea called BeBiDa for Best-Effort Big Data, solely based on RJMS configuration. It makes HPC and Big Data systems communicate through a simple prolog/epilog mechanism, which leverages the built-in resilience of Big Data frameworks while minimizing the disturbance on the HPC workloads.

We present the first study of this approach, using the production RJMS middleware OAR and Hadoop YARN from the HPC and Big Data ecosystems respectively. Note that it can be adapted to any HPC RJMS that features prolog/epilog and any Big Data RJMS that is capable to manage a dynamic set of resources. Our new technique is evaluated with real experiments upon the Grid5000 platform. Our experiments validate our assumptions and show promising results. The system is capable of running an HPC workload with cluster utilization of 70%, with a Big Data workload that fills the schedule holes to reach a full 100% utilization. We observe a penalty on the mean waiting time for HPC jobs of less than 17% and a Big Data effectiveness of more than 68% on average. Our approach is based on configuration and only need 50 lines of additional code to run whether similar approaches are using 100 to 1000 times more code.

## Chapter's Contents

Big Data workload is naturally reaching the HPC environment but the necessary resource management to enable this workload to work properly and efficiently is still in development. As mentioned in this section, there are several existing approaches: from the simplest batch submission to an entire rewrite of the existing tools. A compromise between these extremes can be found with some kind of collaboration between both RJMS to achieve the best of both worlds with minimum hassle. Our approach, described in the following section is aiming this goal.

This chapter is an extended version of the article presented at the IEEE BigData'17 conference [MER+17].

## 4.1 Introduction

As explained in Section 2.1.3, Big Data and HPC convergence is currently happening. However, even though both fields share some objectives, they do not share the same core concepts.

HPC infrastructure can be defined by a set of computing resources with low latency network interconnect, and a Parallel File System (PFS) that provides fast storage for all of these resources. HPC workloads consist of a set of rigid jobs with a strict resource and time requirement (i.e. 10 CPUs for 2 hours). This time requirement is called walltime: it is not a prediction of the execution time but an upper bound after which the job will be killed.

On the other hand, a Big Data workload is made of jobs which can adapt to the number of available resources during the execution. This kind of jobs is called "malleable jobs" in the Feitelson and Rudolph terminology [FR96]. Also, their resource requirements are on multiple resources but not on time (i.e. 4 CPUs and 10 GB of memory). This workload is made of any type of task-oriented jobs that is able to cope with a dynamic resource allocation: Hadoop MapReduce [Fon19b] stands in this category, but also more complex frameworks like Spark [Fon19c] or Flink [Fon19a]. It uses a Distributed File System (DFS) over all the compute nodes to store datasets, using replication to avoid data loss and load balance the data. It comes with the great advantage that most of the data movement can be avoided by bringing computation directly where the data is stored.

Our goal is to provide a way to collocate traditional HPC workloads and Big Data workloads on the same HPC infrastructure, in a way that one user can submit HPC and/or Big Data jobs directly to the RJMS, like he/she is used to do, without modifying the resource managers. We propose a new system called BeBiDa that performs Best effort Big Data Analytics on HPC infrastructure. Figure 4.1 gives an overview of the proposed solution. To get the best of both worlds, we keep the PFS and the DFS leveraging their specialization for each workload: we made the choice to use a DFS to avoid the traditional HPC I/O bottleneck and scale linearly in terms of bandwidth [Xua+15], without adding expensive hardware to scale up the PFS [Gaf+14b].

Collocation is interesting for owners, users and operators of these platforms:

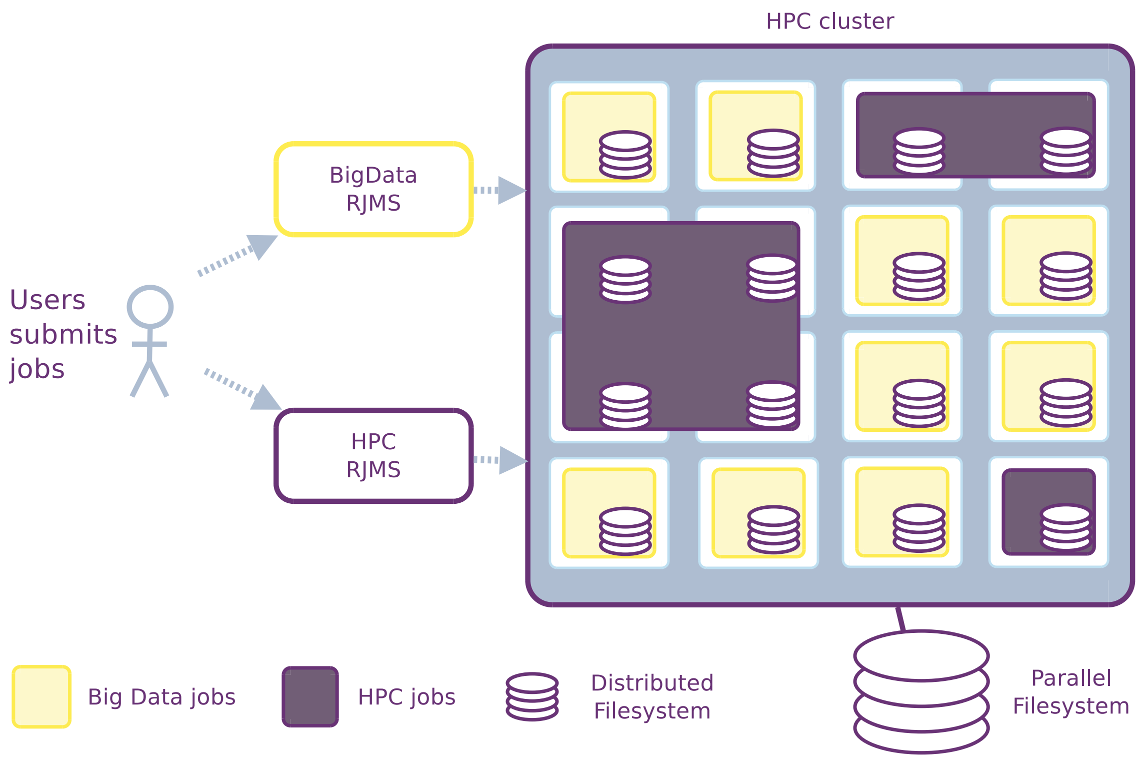


**Figure 4.1.:** BeBiDa system overview: The cluster is capable of dynamically share resources between HPC parallel jobs and Big Data applications. We choose to use a centralized file system for HPC and a decentralized file system for Big Data.

- From the **owners** point of view, collocation increases the platform utilization and attracts new users to HPC clusters by providing Big Data tools alongside traditional HPC tools.

- From the **users** point of view, collocating HPC and Big Data workloads will enable new workflows that mix both approaches and benefit from the best computation tools from the HPC field and the best data analysis tools from the Big Data field.

- From the **operators** point of view, HPC systems, on their own, are producing a large amount of data from hardware counters and services logs. The administrators of HPC systems could take advantage of Big Data tools to do constant analytics of these data in order to detect silent failures, user misuses, configuration errors and so on.

To reach our objective we don't want to provide yet another tool that claims to manage every possible workload while reducing the number of specific features available. We want a solution as simple as possible without losing features. Moreover, RJMSs are massive industrial-grade projects highly optimized for their specific workloads: for instance, Slurm (RJMS for HPC) and Yarn (RJMS for Big Data), have more than 400.000 and 580.000 effective lines of code respectively[1]. That is why we want to fully leverage the advantages of each middleware on their area of expertise.

The contributions of this chapter are:

1. A definition of the HPC and Big Data collocation problem.

2. A new method, called BeBiDa, for collocating HPC and Big Data workloads.

3. An implementation of this method using industrial grade RJMS.

4. An in-depth analysis of this implementation.

The remainder of this chapter is organized as follows: First, we define problems regarding the assumptions listed in Section 4.2. Then, we describe the original idea of simple communication between RJMS with prolog/epilog to make them

[1]Computed with the Cloc software on the official GitHub repository excluding other language than the main one, blank lines, and comment lines

interact with different priority levels in Section 4.4. Section 4.5 provides a proof of concept implementation of our approach and the results of its evaluation. Finally, Section 4.7 concludes this chapter.

## 4.2 Definition of the Collocation Problem

HPC RJMS schedules jobs on computing resources (generally processors). A job requires a fixed number of resources and a certain amount of time: the walltime. If a job runs longer than its walltime, the job is killed. HPC jobs are seen as *black box* by the RJMS: there is no communication between the job and the RJMS.

On the other side, Big Data RJMS schedules applications on a set of resource containers, i.e. a certain amount of computing units and memory. An application is composed of a big number of subtasks with some dependencies (generally a DAG). Each application asks for resource containers with the best data locality available, then the application runtime schedules the application tasks on those containers. The way the subtasks are scheduled on the containers depends on a dialog between the Big Data RJMS and the application runtime. The number of containers allocated to an application is dynamic over time.

We are interested in the problem of collocating a Big Data RJMS and an HPC RJMS, in order to run a Big Data workload ($W_{BigData}$) and an HPC workload ($W_{HPC}$) on the same cluster. Here is a general definition of this problem.

We defined $M$ as the set of all the considered machines (a.k.a. resources). The machine set $M$ is divided into groups defined as follows:

- $M_{HPC-only}$ are dedicated resources for $W_{HPC}$, they are not visible for the Big Data RJMS
- $M_{BigData-only}$ are dedicated resource for $W_{BigData}$, they are not visible for the HPC RJMS
- $M_{shared}$ is the part of the cluster that is shared between the two systems: it can be used to execute jobs from both $W_{BigData}$ and $W_{HPC}$.

- $M$ is union of all these sets:

$$M = M_{HPC-only} \cup M_{shared} \cup M_{BigData-only}$$

To be clear, the two RJMSs do not have the same vision of the resources and only have access to a part of $M$:

- the HPC RJMS is only aware of $M_{HPC}$:

$$M_{HPC} = M_{HPC-only} \cup M_{shared}$$

- The Big Data RJMS only sees $M_{BigData}$:

$$M_{BigData} = M_{BigData-only} \cup M_{shared}$$

Note that if $M_{shared} = \emptyset$ we go back to a static partitioning of the cluster. On the other hand, if $M_{BigData-only} = \emptyset$ and $M_{HPC-only} = \emptyset$ then, the cluster is entirely shared.

Also, to be able to talk about HPC jobs and Big Data applications seamlessly, in the rest of this chapter, the generic term *jobs* will be used for both.

For the sake of clarification, the following list presents the assumptions that are used for the rest of this chapter:

- The cluster contains a set of machines $M$
- One HPC RJMS is set up, and have access to $M_{HPC}$ to schedule jobs of from $W_{HPC}$.
- One Big Data RJMS is set up, and have access to $M_{BigData}$ to schedule jobs from $W_{BigData}$.
- All the software stack is managed by an administrator and not by the users. Both systems are already configured to start their jobs: no deployment needed.

- Jobs in $W_{HPC}$ use a dedicated Parallel File System (PFS) to store data.
- Jobs in $W_{BigData}$ use a Distributed File System (DFS) that is spread over a set of machines:
$$M_{DFS} \in M_{BigData-only} \cup M_{shared}$$
- HPC Jobs are statically allocated (rigid or moldable [FR96]).
- Big Data jobs are malleable and resilient to the interruption of all, or part of, their resource allocation.

## 4.3 Related works

Some of the most famous resource managers for Big Data are Corona [Fac12], Mesos [Hin+11], Omega [Sch+13], and Yarn [Vav+13a]. A comparison of these tools can be found in [Vav+13b]. They are all maintained and used by a big web company for their daily data analytics jobs. On top of these resource managers, frameworks help users to build and run their applications. Most of these frameworks support at least the MapReduce paradigm [DG08]. Specificity of the Big Data workload is its malleability due to the fact that the application work is decomposed into small tasks that can be easily spread on resources. Some of the most known frameworks are Spark [Zah+12], and Flink [Car+15]. To enable dynamicity and resilience, these frameworks are connected to the RJMS by an API. This connection permits the framework and the RJMS to negotiate resources acquirement dynamically even during the jobs executions.

In the HPC community, some notable resource managers are Slurm [YJG03], Torque [YJG03], PBS pro [NSJ04], and OAR [Cap+05]. The main difference with Big Data resource managers is that they manage traditionally jobs as “black boxes”; there is no communication between the job and the RJMS. As a consequence, the way HPC RJMSs traditionally manage jobs is rigid. A job can rarely grow or shrink in size. Another difference is the management of time: for each HPC job a walltime should be given and resources reservations are common. These computation frameworks that exist in the Big Data field have no direct equivalent. To achieve the best performance for each application, HPC frameworks are not using high-level paradigms, but low-level message passing libraries. For example, MPI can be used to implement a wide range of different high-level paradigms, even MapReduce [PD11].

At the application level, the Big Data software stack is being transformed to take advantages of the high-performance hardware provided in HPC, like fast interconnect and fast Parallel File Systems (PFS) [Was+15]. Some are taking the opposite direction, and use HPC traditional technologies, like MPI, to build new Big Data tools because the resilience of the Big Data software stack does not compensate for the lack of performances [ROA15b]. Thus, we conclude that it is worth to use the traditional Big Data tools on HPC infrastructure only if we leverage their dynamicity and resilience.

More than a decade ago, when desktop grids and volunteer computing have been introduced, systems such as Boinc [And04] and Ourgrid [And+03] have started handling collocation of different types of workloads using similar techniques like ours. In the same area, the lightweight grid meta-scheduler Cigri [GRC07] is based on OAR's RJMS best-effort jobs type in order to schedule bag-of-tasks grid workloads when the local HPC jobs leave idle resources on the cluster. BeBiDa mechanism is based on the same idea of exploiting idle resources but without introducing another level of scheduling.

### 4.3.1 Static partitioning

At the resource management level, it exists multiple ways to make Big Data and HPC workloads run together. The most obvious one is to have two different clusters, or split a cluster into two parts, where each part is dedicated to one workload: A part of the cluster is chosen to host a Big data analytics framework (BDAF) and the rest is allocated to the HPC workload. But, data transfer between the two parts will be a serious bottleneck. Moreover, there is no load balancing between the partitions due to this strict separation of concerns. This is not satisfactory in terms of resource management because of the lack of resource sharing: it leads to resource underutilization as it is shown in Figure 4.2.

### 4.3.2 Batch submission

The idea of the batch submission is to ask for resources to the HPC RJMS, like a classic HPC job, and then install and run Big Data applications inside these

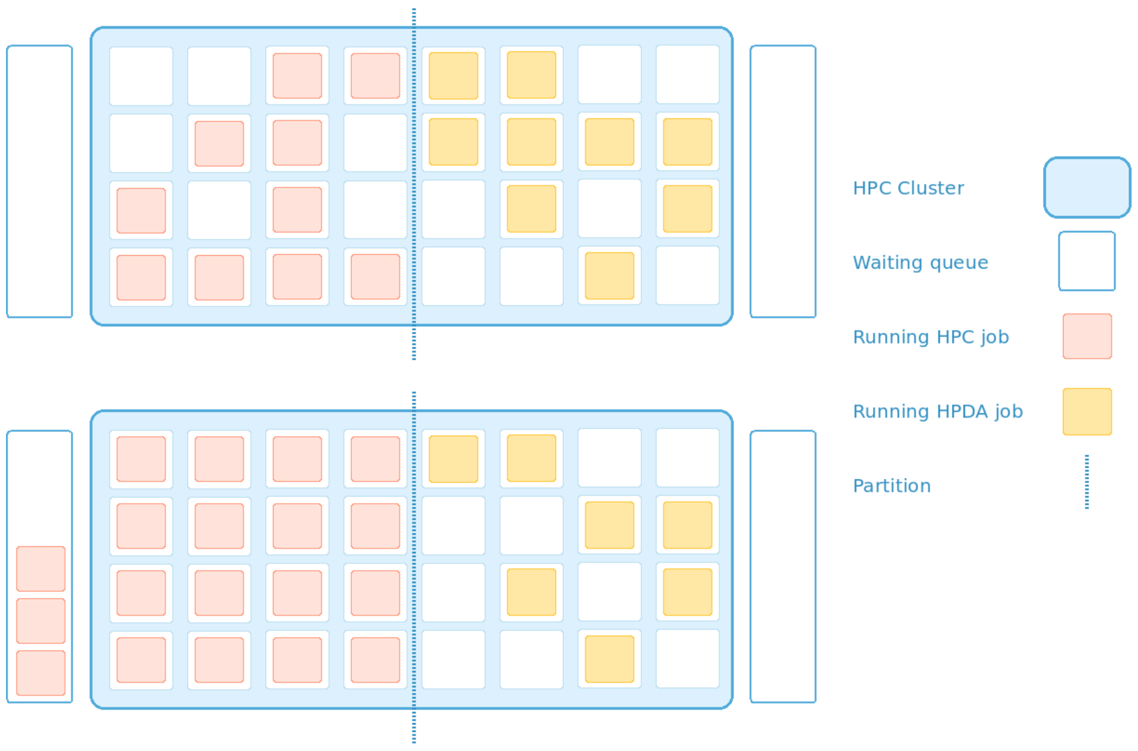


Figure 4.2.: Resource utilization using static resource partitioning approach leads to a waste of resources: In this example, the HPC queue has jobs that are waiting but cannot be scheduled on the Big Data side.

allocated resources. It can be done as a normal user or using a modified HPC RJMS.

### As a user

A simple approach is to let the user manage his own Big Data software stack inside HPC batch jobs using a set of scripts [Chu13]. It works for simple workflows with a small amount of data but it has drawbacks: (a) Even if the scripts can help, the user has to operate and configure part of the system by himself and it can be complex while the software stack grows. (b) The Big Data software stack has to be deployed, configured, started and shutdown for each job: the overhead is important, especially for small jobs. (c) If the amount of data is too big to fit in the user home, third-party storage would be needed and the data movement produced by data ingestion can dominate the execution time [KTB11].

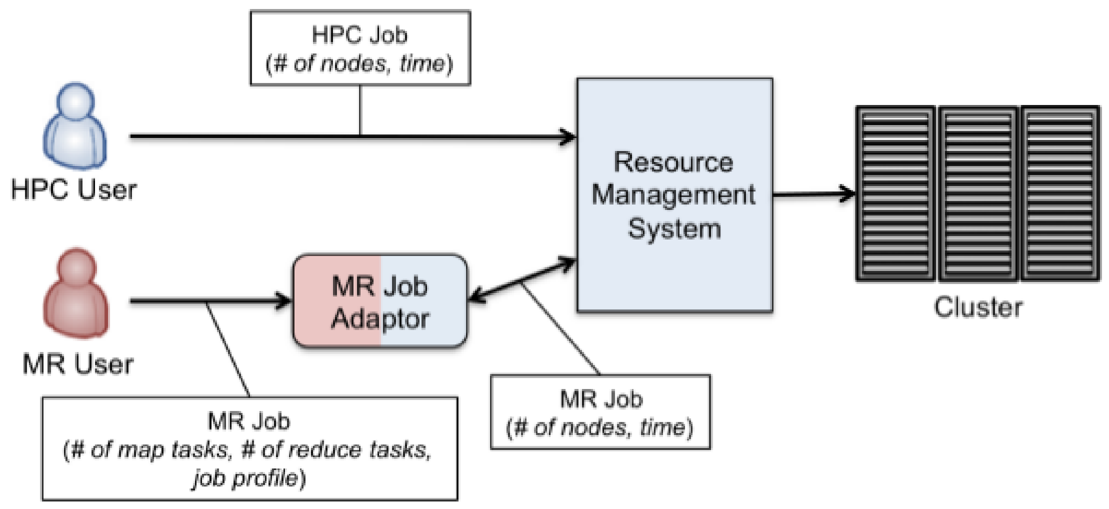


Figure 4.3.: Resource sharing with adapter. This Figure was taken from [NFD12]

#### As a RJMS plugin

A way to provide this kind of feature is to add a plugin to the HPC RJMS to allow the user to run his Big Data application directly from the HPC RJMS interface: All the configuration, installation, and tear down is done by an RJMS plugin.

In [Kas+15] this kind of approach is proposed on top of the LSF scheduler and YARN. It also provides an API to launch Big Data jobs without interacting with the HPC RJMS traditional interface. Except for the (a) issue, this approach has the same problem as the previous one concerning the deployment overhead (b) and the file system (c).

### 4.3.3 Scheduler Adapter

An integrated approach is to add a new abstraction level to integrate one system to the other with an adapter that converts Big Data RJMS resource requirements into HPC RJMS allocations [NFD12; Int17]. This mechanism is depicted in Figure 4.3.

The main issue of this kind of approach is that it is tightly coupled to specific technologies, have to evolve with them and is limited to them. It restricts this solution to the implemented adapter, and the cost of maintaining this whole new layer can be too high.
In comparison, our approach is based on configuration and the only code that has

to be adapted while changing technologies is the prolog/epilog scripts that count 50 lines of bash in our implementation available in Annex A.1.

### 4.3.4 Two-level scheduling

The third way is to use another level of scheduling to achieve a dynamic partitioning of the available resources on the cluster. Thus, the Big Data and the HPC traditional workloads can run side by side on an HPC cluster supervised by a Meta scheduler.

#### Using a third tool

The use of a meta-scheduler (Two-level scheduling) like Mesos [Apa15] can be a good approach to provide dynamic resources sharing between the RJMS and the Big Data schedulers. However, the HPC RJMS must implement the Mesos framework API in order to dynamically manage its resources. Also, most of the HPC scheduling heuristics are optimized for a static set of resources, thus the scheduling performance of classical RJMSs may dramatically reduce because it has to constantly re-schedule depending on the resource obtained by Mesos.

#### Implement meta-scheduling capabilities in the RJMS

Univia choose to implement is their RJMS (Grid Engine) the server side of the Mesos Framework API [Pri15]. Thus, it can manage all Mesos compliant frameworks and applications [Apa15] over a mixed environment (Grid) that may include HPC.

This flexibility comes with loss of control. In general, if a RJMS have meta-scheduling features that allows it to delegate a dynamic part of its managed resources to a BDAF, it can take advantages of the BDAF data-centric scheduling. However, by doing this the RJMS is not able to implement fine-grained scheduling policies, as it is not aware of what is happening on the delegated parts of its resources.

**Pilot-based abstraction**

One more idea would be to run HPC jobs within a Big Data stack. The bigger problem with this approach is that most HPC applications are not able to run within Big Data RJMS because they need to implement a communication layer with it. A rewrite of most HPC applications is required which makes this approach unfeasible in the HPC community. In a recent study [Luc+16], Lucow et al. propose to solve this problem by encapsulating both HPC on Hadoop and Hadoop on HPC with a Pilot-based abstraction. The authors provide a new layer of software that glue the two systems. It allows HPC users to use the HPC-based mode when it is required by the application (or by the platform) and the Hadoop based mode in the case of a hybrid platform.

The main reason we take another approach is the complexity of this solution. Also, it relies heavily on Hadoop while our approach is more technology agnostic and capable to adapt to any kind of system that is able to dynamically commission and decommission resources and any HPC scheduler that has a prolog/epilog mechanism.

### 4.3.5 Discussion and Summary

The Table 4.1 summarizes the characteristics of each approach, including the one presented in this chapter. We can observe that, with the added constraint of priority to the HPC workload, BeBiDa is providing better or similar level of features than the other approaches but with 100 to 1000 time less code.

| | **Static partition** | **Deployed in an HPC job** | | **RJMS adapter** | **Two level scheduling** | | | **Bebida** |
|---|---|---|---|---|---|---|---|---|
| | | As a User [Chu13] | RJMS plugin [Kas+15] | BDA over HPC [Int17] | Third tool as master [Apa15] | HPC RJMS as master [Pri15] | Pilot-based abstraction [Luc+16] | Section 4.1 |
| **Resource sharing** | None | Static | Static | Static | Dynamic | Dynamic | Dynamic | Dynamic (with priority to the HPC workload) |
| **Redeploy for each job** | No | Yes (done by user) | Yes (done by the RJMS) | No | No | No | Yes(mode I), No(mode II) | No |
| **Data locality** | Only in the BDA partition | No | No | No | Yes | Yes | No(mode I), Yes(mode II) | Yes |
| **Extra data movement (with separated PFS/DFS)** | Between file system | Manual data staging for each job | Manual data staging for each job | Automatic data staging for each job | No | No | No | No |
| **HPC RJMS requirements** | None | None | Plugin for auto deploy | Programmatic API to submit jobs | Client API to talk to the master | Server API to talk to the clients | SAGA API | Prolog/epilog mechanism |
| **BDA RJMS requirements** | None | None | None | Adapter to convert users' requests | Client API to talk to the master | Client API to talk the master | Only compatible with YARN | Dynamic resource management |
| **Seamless for HPC users** | Yes | Yes | Yes | Yes | No (loss of predictability) | Yes | Yes(mode I), No(mode II) | Yes |
| **Seamless for BDA users** | Yes | No (manual deployment) | No (interact with HPC RJMS) | Yes | Yes | Yes | No(mode I), Yes(mode II) | Yes |
| **Extra requirements and constraints** | No resource sharing between the partitions | BDA stack installable by the users | Need to maintain plugin and BDA images | Need to maintain the adapter | Need to maintain third tool + RJMS APIs | Need to maintain modified RJMS | Need to maintain Pilot agent for both mode | No SLA on the BDA workload |
| **Extra lines of code** | None | ~20k - Shell scripts (Magpie) | Not provided (Proprietary) | ~5k - Java (HAM) | ~30k - C/C++ (Mesos) | ~50k - C/C++ and Python (URB) | ~50k - Python (RADICAL SAGA+Pilot) | 0.05k (only 50) - Shell (Bebida) |

Table 4.1.: Comparison between HPC and Big Data RJMS collaboration approaches. Note that the BeBiDa approach uses several orders of magnitude less code.

To the best of our knowledge, the approach presented in this chapter is the first to use two unmodified RJMSs that collocate HPC and Big Data workloads on the same cluster, without adding a third tool to coordinate them.

## 4.4 BeBiDa solution description

### 4.4.1 Description

The main idea of BeBiDa is to run two industrial grade RJMSs collocated on the same cluster: One Big Data RJMS and one HPC RJMS. Big Data is running mainly in best-effort mode, i.e., it uses **HPC idle resources** as a **dynamic resource pool**. It relies on the prolog and epilog mechanisms that are provided by most HPC RJMSs: The prolog (resp. epilog) is a script that is executed before (resp. after) the execution of each job. The following is a detailed description of this process, as shown in Figure 4.4:

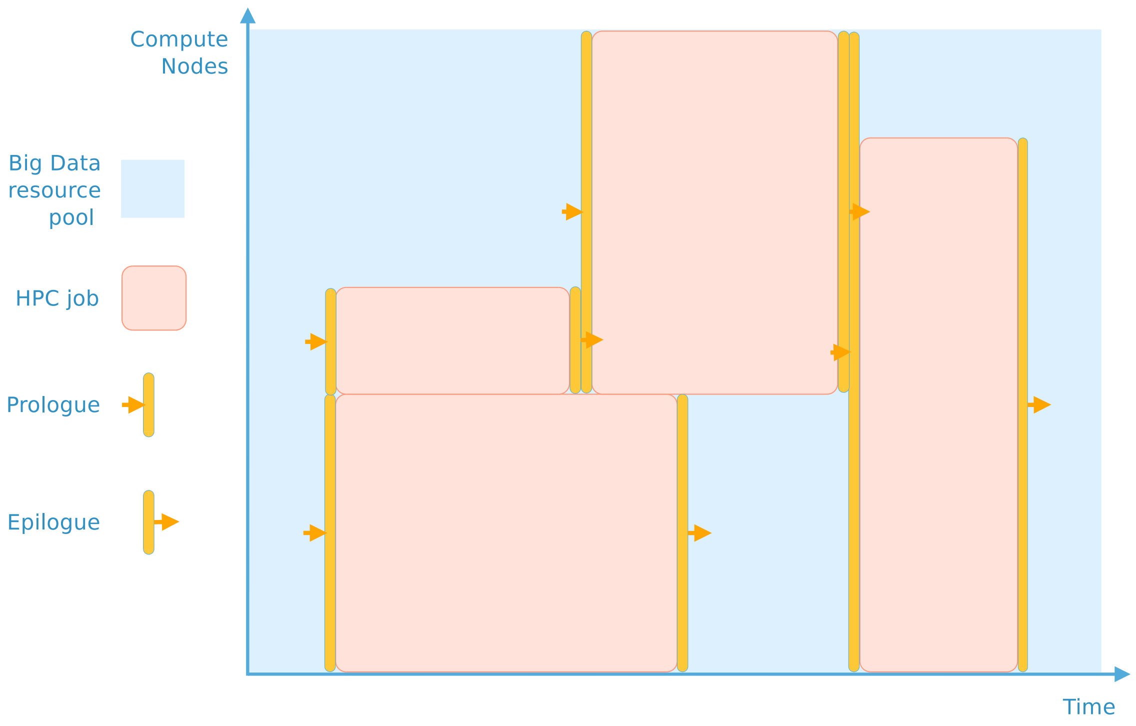


Figure 4.4.: BeBiDa sharing resource mechanism is based on HPC prolog and epilog.

1. By default, all resources are attached to the Big Data resource pool.

2. When an HPC job starts: the prolog decommissions the resources needed by the HPC job from this pool.

3. When an HPC job finishes: The epilog is giving the resources back to the Big Data resource pool.

From our solution derives the following additional assumptions:

- $W_{HPC}$ has priority over $W_{BigData}$.
- The set of collocated machines $M_{shared}$ are seen by the HPC RJMS as always available.
- The Big Data RJMS sees only the part of $M_{shared}$ that is not running any HPC jobs as available. This part is dynamic over time.

### 4.4.2 Advantages and drawbacks

The main advantages of this solution are to be easy to implement and as simple as a configuration problem for both RJMSs: it does not require any third tool, it avoids tight coupling between the two systems and can be easily implemented into most of the HPC and Big Data RJMSs.

This configuration has to be done by the cluster administrator that manages both RJMSs. The solution is transparent to final users: they submit HPC and Big Data jobs through there respective traditional interfaces.

Only small scripts are needed (50 lines of bash available in Annex A.1.) for HPC RJMS prolog and epilog, no code modification is needed for any tool inside both software stack.

Also, it leverages Big Data frameworks resilience and dynamicity by using a dynamic resource pool and does not disturb the HPC applications by ensuring that no processes are left on the compute nodes after decommissioning, except for the DFS daemon. The impact of this daemon has been studied more thoroughly in another context [Hie18] and discussed in Section 2.5.2.

It also impacts HPC jobs' waiting time due to the prolog/epilog execution time: this is discussed in Section 4.5.2.

For the Big Data workload, resource preemption during a job execution can be a problem: if it happens too often, its cost can be unsustainable and Big Data computation will become unpredictable. This is also measured in Section 4.5.2.

Obviously, this approach will perform well when Big Data is dominant in the workload; i.e., when the HPC utilization is low. On the opposite, we expect to have poor Big Data performance on a loaded HPC cluster because the number of killed "on duty" workers will increase, inducing more re-computation and data movement.

## 4.5 Experimentations with BeBiDa

Our goal in this section is to describe our BebiDa proof of concept implementation and to present the experiment design and results.

### 4.5.1 BeBiDa implementation

In this implementation, no modifications have been done to the RJMSs. Only their configuration has been adapted to fit our needs. Configuring industrial grade software can be seen as a straightforward task but it is not. Both RJMSs have hundreds of parameters that needed to be set (scheduling, resources, accounts, security policies, file systems, ...). All code and experiment scripts can be found in a public Git repository[2]. These experiments are implemented using State-of-the-Art tools for reproducibility [Rui+15][Imb+13].

#### Hardware

Our experiments were carried out on Grid5000 [Bal+13]. We chose the Graphene cluster on the site of Nancy: It has 1x Intel Xeon X3440 with 4 cores/CPU, 16 GB RAM and 298 GB HDD. We select $M = 34$ nodes, with $M_{shared} = 32$ and $M_{BigData-only} = 2$ and an additional master node to host YARN and OAR masters.

[2]`https://gitlab.inria.fr/mmercier/bebida`

### Software

We used OAR 2.5.4 as HPC RJMS. OAR is easy to deploy and configure and integrates well with our testbed.

OAR default configuration is kept: It uses a conservative First Come First Served (FCFS) scheduling policy with backfilling.

Its allocation policy, i.e., how the scheduler chooses between available slots for a job, is *smallest resource ID first*: that produces some kind of "gravity" for the job placement. The prolog (resp. epilog) script is simple: it gets the resources of the job to be launched (resp. ended) and removes (resp. add) its resources from Yarn. The scripts wait for the resources to be removed from (or added to) Yarn before ending.

Prolog and epilog scripts are added as hooks that are executed before and after each HPC jobs on OAR's configuration.

We choose YARN from Apache Hadoop 2.7.3 as the Big Data RJMS. With Yarn, we use Spark as a computation framework and HDFS as a distributed file system.

This is one of the most used Big Data software stack, and Yarn has the capacity to dynamically change the number of resources used with minimal disturbance on the workload. We used the capacity scheduler which is the default scheduling algorithm for Yarn in the Apache distribution. The minimum resource container allocation is 1 core and 1 GB of memory and the maximum is the whole node capacity.

Spark 2.1.0 is configured to run 2 executors by nodes and all the applications are run in client mode with 1 core and 1 GB of memory for the application master.

As we want Spark to run in a very dynamic environment, we set both parameters `spark.task.maxFailures` and `spark.yarn.max.executor.failures` to 250 to prevent jobs from getting killed because of resource preemption.

Also, because the Spark Application Master is a single point of failure for the application, we use the node label capabilities of the YARN capacity scheduler and of Spark to pin the Application master to the $M_{BigData-only}$ nodes. As a side effect, this setting limits the number of concurrently running applications to the number of Application Master $M_{BigData-only}$ can host. ,

### Workloads

For this experiment we need mixed workloads of HPC and Big Data jobs. However, because such workloads are pretty new, we do not have access to real traces for now. To be able to test our system on different HPC and Big Data workloads we chose to generate those workloads based on the statistical study on HPC workloads done by Feitelson[Fei15a]. In order to generate these workloads we need different kinds of applications, called job profiles, to populate each type of workloads.

The Big Data workloads are composed of 3 different types of applications taken from the BigDataBench benchmarks [Wan+14]: Grep, WordCount, and K-means. We have use 3 sizes of datasets for each application (32 GB, 64 GB, and 128 GB) generated using the BigDataBench generation tool [Min+13]. These datasets are injected into HDFS at the beginning of the experiment with a replication factor of 3.

Big Data workloads are generated with parameters that make most of the submissions at the beginning of the workload to be sure that there are enough jobs to fill the available resources from the beginning of the execution.

The HPC workloads are composed of 3 different applications type taken from the NAS Parallel Benchmarks [NAS16]: IS, FT and LU. Those applications are compiled for different number of resources, each power of two from $2^2$ to $2^7$ (the number of cores in the cluster), and different size from C to E. These job profiles are filtered to obtain 14 jobs profiles that run for at least 1 minute and less than half an hour. The workloads generated contain 250 jobs randomly picked in this pool using the previously mentioned method.

We have run simulations using Batsim [Dut+16] in order to select the generation parameters that result to an average utilization of around 70%. We have selected this level of utilization because it is representative of small to medium size HPC center.

### 4.5.2 Results

We evaluate our solution based on how well both systems, HPC and Big Data, coexist. Thus, we have measured the throughput of Big Data jobs during the execution of the HPC workload, and the HPC workload mean waiting time with and without a co-located Big Data workload. By comparing both results, we are able to measure the impact of the Big Data workload on the HPC system.

With this experiment, we want to answer the following questions:

1. Does BeBiDa work in real conditions?
2. What is the overall utilization of the cluster?
3. Is Big Data computation effective in these conditions?
4. What is the impact on the HPC workloads?

First, the system works correctly with the aforementioned configuration and setup: It is able to run HPC jobs normally and Big Data jobs in the holes of the HPC scheduling. You can see a Gantt chart of one experiment instance in Figure 4.5. We try to pick a representative experiment regarding HPC utilization. Note that in the Big Data workload each band represented a Spark executor with a common color for each application. Spark is configured to have 2 executors per node and when this is the case the executors are overlapping and the colors are mixed. You can also notice that, as the prolog/epilog scripts are very simple, they do not have a vision of what will happen in the future: that's why even if two HPC jobs are consecutive, executors are started during the guard time of 60 seconds that OAR puts between 2 jobs. Also, we can notice that executors are overlapping on HPC jobs: this is due to the fact that Spark induces a delay between the real killing of an executor and the report of this kill in the application logs.

Having a full cluster utilization is one of the HPC owner main goals. With BeBiDa, The overall utilization of the cluster virtually goes from 70% on average to 100% if the $W_{BigData}$ contains enough work to fill the holes left in the schedule by $W_{HPC}$. But, this statement is mitigated by 2 points: first there is a small delay when a resource goes back to an idle state before it is actually used to compute $W_{BigData}$ which correspond to the Big Data scheduler's delay; second, resource preemption

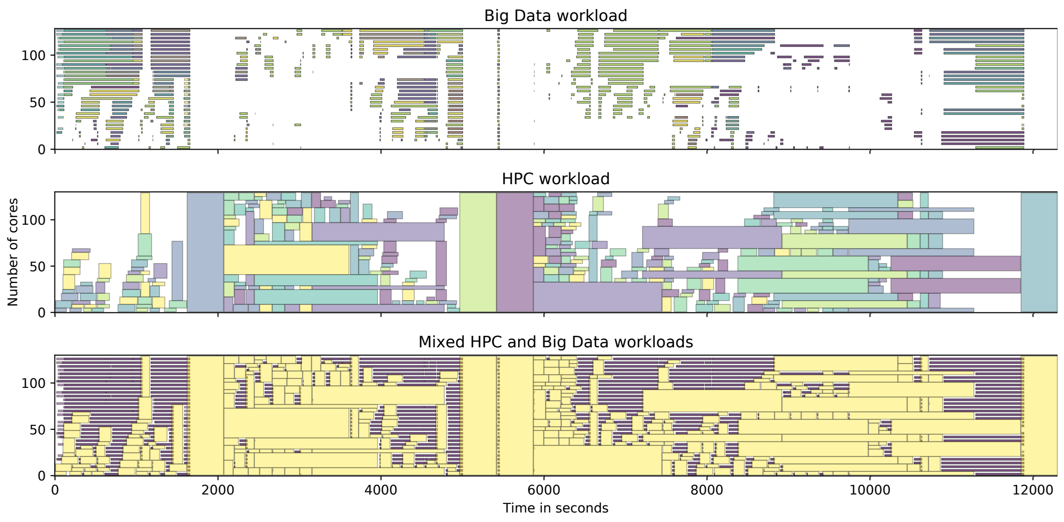


Figure 4.5.: Example of one experiment with BeBiDa enabled

cost on the Big Data jobs reduces their work efficiency. The cost of this preemption highly depends on the computation framework capacity to manage these events. To tackle this last point we define the efficiency $E$ of the computation done by the Big Data framework, here Spark. It can be computed with the time spent in tasks that are complete and not resubmitted over the total task computation time:

$$E = \frac{\sum T_{complete} - \sum T_{resubmitted}}{\sum T_{complete} + \sum T_{failed}}$$

To understand this metric the reader needs to understand how Spark works: It uses an internal in-memory representation called RDD [Zah+12] that uses lazy evaluation and lineage to recompute only what is necessary if some intermediate data is missing. When a preemption happens, currently running tasks are killed; computation time is lost ($T_{failed}$), and these results are lost but also the intermediate data of previously successful tasks have to be recomputed elsewhere: this is the time taken by resubmitted tasks ($T_{resubmitted}$). Figure 4.6 shows that the effectiveness goes from 44% to 91% with a mean of 68%.

We also compute data locality for each task i.e., the fact that data processed by a task is fetched from the local memory or from a distant node. At 71% on average, task and data are located on the same node (NODE_LOCAL), but only 10% of the overall tasks are processed on the same executor (PROCESS_LOCAL). It means that most of the data is accessed inside the node memory but, as there are 2 executors per node, it happens that only a small part of data accesses are done on executor process memory directly.

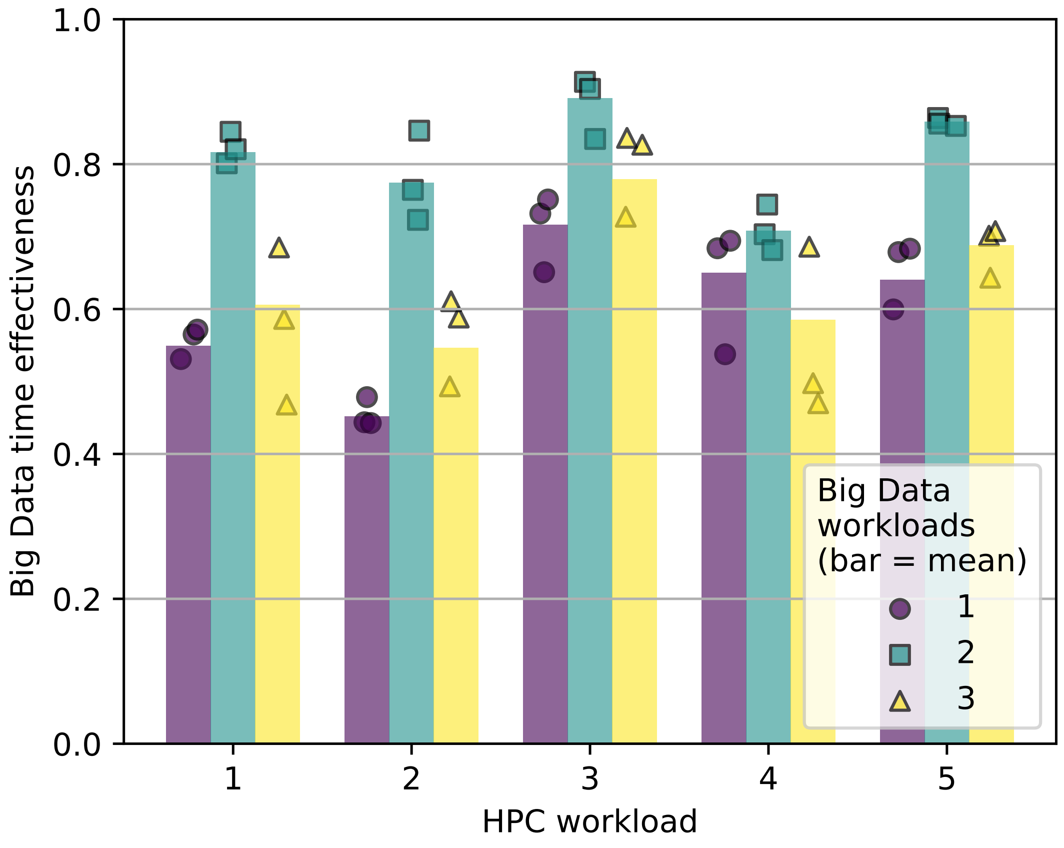


Figure 4.6.: $W_{BigData}$ Time effectiveness $E$ regarding HPC workload utilization. Minimum is 44%, maximum is 91%, with a mean of 68%.

We also measure the time taken by the prolog and the epilog. The prolog is triggered when an HPC job is allocated: it decommissions the resources associated with this job by killing YARN containers if present, and the NodeManager afterward on each resource. This process (i.e., the prolog) takes from 2 to 16 seconds depending on if there is running containers or not: It takes around 16s if there is 1 or multiple containers. The epilog process that restarts the NodeManager takes less than 3 seconds.

Both hooks imply a small penalty on each job's waiting time that can be observed in Figure 4.7. But, The mean waiting time overhead is less than 17% on average.

We can notice that, for some measures, the workload 3 mean waiting time is highly impacted. This is due to a bad backfilling decision on a big job that delayed a lot of small and medium jobs. It only happens in specific conditions that are not met

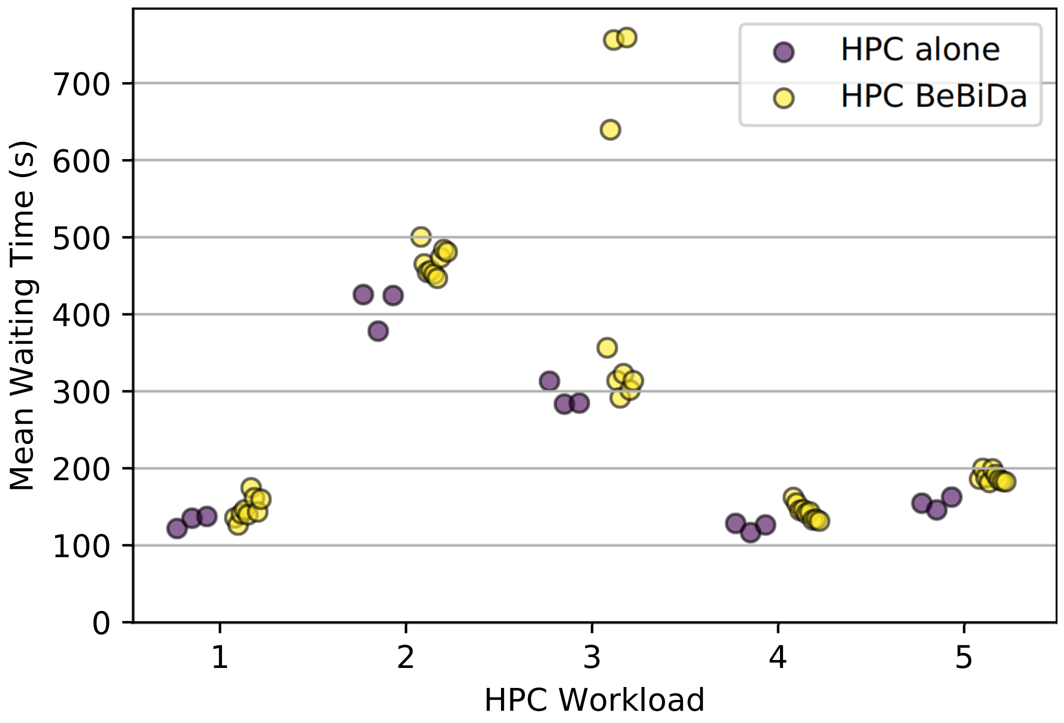


Figure 4.7.: Overhead on the HPC workload $W_{HPC}$ in mean waiting time.

at every run. It illustrates the highly sensitive nature of scheduling where small changes can dramatically impact the whole system.

All experiments were executed 3 times.

The mean execution time of HPC jobs is also impacted by BeBiDa. It is increased by 6% on average. This is due to the network contention created by $W_{BigData}$ and the memory and computation overhead due to the HDFS daemon that is still running during HPC job execution.

## 4.6 Discussion and perspectives

We describe in the previous chapter an implementation of the BeBiDa methodology. As we have seen that its performances are acceptable but the implementation have shortcomings. In this section, we discuss how they can be overcome, and the new research challenges that they open.

### 4.6.1 Improve RJMS collaboration

We need better guarantees on the quality of service for the Big Data workload inside a BeBiDa setup. Even if there is no room in the HPC schedule because of high utilization of the cluster due to the HPC workload, the Big Data RJMS should be able to drain its waiting queue.

A promising approach is to create a simple tool, called Punch, that will create holes in the HPC RJMS scheduling simply by submitting an HPC job in a high-priority queue, but leave the acquired resources to the Big Data RJMS.

To do so, Punch would listen to the Big Data RJMS queue state, and when the queue is too loaded i.e., exceeds a threshold, it will submit a job that does not trigger the BeBiDa prolog/epilog: Consequently, the resources associated with this job would be left in the Big Data RJMS resource pool, and give room for the waiting Big Data jobs to be executed.

### 4.6.2 Preemption avoiding

We would like to avoid killing Big Data jobs as much as possible. Undertaken methods are detailed in the following sections. Unfortunately, these approaches are currently not tested because of the experimentation unaffordable cost.

#### Blind methods

A possible optimization is to give a static number of resources to the Big Data with weak guarantees: it means that those resources can be preempted only in certain conditions, like a threshold on the HPC utilization level or on the job resource size to minimize the impact on the HPC workload.

It is possible for the HPC RJMS to take smarter decisions even if it is blind regarding the other system that runs on idle resources. Even without knowing about the running Big Data jobs, the HPC RJMS placement algorithm can at least try to be smart by using some simple heuristics to avoid killing jobs. For example, when the RJMS has to take a scheduling decision about a new job and it has different placement possibilities we can order these possibilities regarding different policies:

Random, youngest first, oldest first, ordered by resource ID, . . . Youngest first, using the free slots that were released the most recently, seems the more promising one because it will kill the workers that are available for Big Data for the shortest time, thus it has less probability to kill a long-running Big Data job.

Smarter policies can also be developed in order to take into account both workloads' specificities. For example, topology aware allocations are critical for HPC jobs (as most jobs depend on the network performances), while Big Data applications tend to have more allocation possibilities thanks to data replication. Thus, a policy can be developed that will force HPC job to have always near-optimal allocation, at the cost of a lower HPC utilization. The lower HPC utilization would be compensated by the Big Data throughput increase.

From the Big Data point of view, there are multiple possible optimizations, but they depend on the computation framework properties. In the case of Spark, it is possible to minimize the cost of resource preemption by raising the persistence of the RDD: The simplest way is to increase the RDD replication factor. This should reduce the number of resubmitted tasks because intermediate data are kept in different resources that may not be preempted at the same time. Moreover, to avoid re-computation of the intermediate data lost during preemption, it can be envisioned to modify Big Data applications to force intermediate data persistence.

#### Explicit collaboration

We would like to minimize preemption cost for Big Data applications by giving to the HPC scheduler insight to avoid preemption of resources that actually run Big Data tasks. Indeed, the pool of resources managed by the Big Data RJMS may not be fully used and preempted resources can be idle. One way to achieve this is to make the Big Data RJMS submit best effort jobs to the HPC RJMS when it allocates resources containers, so the latter can try to avoid killing them with taking this information into account in his allocation policy. HPC scheduler can even be informed of the jobs' interruption cost regarding computation stage. For example, with MapReduce it is better to stop a Map in its early phase than to stop it during its shuffling phase because all the intermediate data would be lost. This kind of improvements implies to make information transit between the two systems. The prolog/epilog scripts can be improved to gather more contextual data and provide it to the HPC RJMS.

Another approach could be to minimize the number and the cost of preemption by giving to the Big Data scheduler a vision on when a resource will be preempted. With this kind of information, the scheduler can decide to migrate tasks and avoid allocation that will be preempted soon. It also permits to avoid the HPC starting time overhead by removing resources from the Big Data pool earlier. The problem is that backfilling can disturb this vision of the future. It also reaches the problem of Big Data task execution time prediction because this information is required for the Big Data framework scheduler to make good decisions. Of course, this mechanism comes with the additional communication cost between the two RJMS (greater than the simple resource preemption action).

Explicit collaboration should be avoided because the implementation of such collaboration would break one of our requirement: Do not modify existing RJMSs.

### 4.6.3 Data Management

In the HPC community Parallel file systems (PFS) (like Lustre, GPFS, BeeGFS, . . . ), have been developed to ease data management while improving the performances. The major problem of this approach is the centralization that creates a bottleneck, and it is already a problem in HPC infrastructure [Use+10]. That is why we choose to dismiss the use of the PFS for the Big Data workloads.

In the Big Data field, this bottleneck is avoided by using a distributed file system (DFS): computation is brought to data and not the other way around. Also, data replication do not provide only resilience but data balancing over the cluster. It induces a new problem which is data locality. In our context, the data locality matter is complex because of the HPC workloads which dynamically hinder the use of part of the resources. This is left for future works.

Whatever the distributed file system used, we can try to optimize different aspects:

For example, trying to make the DFS daemon not persistent on compute node reserved by an HPC job to minimize the impact on HPC workload. The problem is, for recreating the lost replica, the DFS replication balancing mechanism will generate a lot of communications. To avoid this without modifying the DFS we can toggle or tweak the `safemode` time (the security time between a failure detection and the rebalance of the replica). We also can change dynamically the number of replicas that can be set for each chunk of data. Another option is to modify (or create a new) DFS to make it capable of managing this situation.

If we stick with a persistent DFS daemon, we expect some disturbances in the HPC jobs computation due to the daemon itself, and on the communications due to data movements. For the computation noise, it can be reduced by limiting the resources available to the daemon using Linux kernel technologies (cgroups, CPU pinning, hyperthreading, . . . ), or the by the reservation of one core. For the network interference, it can be limited by using Quality of Services technologies available on both Ethernet and Infiniband networks. Also, it can be avoided on a cluster that is equipped with two types of networks: we can assign the DFS traffic to Ethernet and keep the Infiniband network dedicated to HPC workload.

### 4.6.4 Time Fragmentation

With the experiments done for this chapter, we find out that the level of utilization is not correlated to the effectiveness of the Big Data workload. It mostly depends on the application possibility to run a complete stage before being preempted. This is hard to measure and even harder to find a metric that represents it. But, if we had this kind of metric we would be able to adapt the HPC scheduling to minimize the number of preemption without any knowledge of the other workload. This can even lead to a generalization of the concept for Cloud Computing IaaS system minimizing virtual machine migration, Power Management minimizing node extinction, or any kind of mechanism that manages a dynamic resource pool.

The notion of "time fragmentation" that characterizes the free resource's distribution over time is the closest metric we found in the literature [GS09]. This metric $F$ is defined using the list of empty time slots $f$ for each resources $i$ in $[1, n]$. The $p$ parameter is used to diminish the importance of the small free slots, and respectively increase the importance of the large ones.

$$F = 1 - \frac{\sum_{i=1}^{n} f_i^p}{(\sum_{i=1}^{n} f_i)^p}$$

But this metric gives time fragmentation only in one dimension i.e., time per resources, and do not take into account multiple resources holes. In fact, fragmentation definition depends on the shape of the wanted free slots and have to be defined regarding to the use case: e.g., Big Data analytics jobs, On-demand virtual machine or containers for short or long-running services. Each of these workloads has different scheduling constraints. For example, long-running services may require fewer resources but need to run on large time slots to avoid costly migration and keep a certain level of service.

Finally, a more adapted metric remains to be defined, and we think that it should be not only one metric but one for each use case, that fits each use case constraints.

## 4.7 Conclusion

This chapter defines the problem of collocating HPC and Big Data workloads on the same HPC cluster. We propose a new approach called BeBiDa that is seamless for end users and requires only configuration from the cluster administrator. It is based on a simple job prolog/epilog mechanism, which is very common on HPC batch schedulers, and on the capacity to manage a dynamic set of resources which is standard for Big Data RJMSs. This solution exposes all nodes are not used by HPC jobs to the Big Data resource manager as a dynamic pool of resources.

We provide a proof of concept and run experiments over it. It shows that with an HPC utilization of around 70%, the system is able to run Big data jobs on the unused resources. The high dynamicity of the Big Data resources pool induces a loss of efficiency. We define the time effectiveness metric to measure this efficiency. In our experiments, the time effectiveness goes from 44% to 91% (with a mean of 68%) depending on the Big Data and the HPC workloads. A key point of the BeBida approach is that it is based on configuration and only requires 50 additional lines of codes, where similar approaches need from 5000 to 50000 lines.

In Chapter 5 we present a model of this system over the Batsim simulator to be able to explore a wider range of parameters, and study the trade-off between PFS and DFS for this kind of hybrid workload. Our approach can be extended to any system with the notion of scheduling with priority. Finally, finding good metrics that can be optimized in new scheduling heuristics remains a big challenge.

# I/O Aware Simulation of HPC and Big Data Converged Platform

5

## Chapter's Abstract

Experimenting on converged Big Data and HPC infrastructures like the one proposed by the BeBiDa approach requires the use of simulation. However, existing infrastructure simulators are mostly tied to a specific tool and not able to simulate the I/O behavior of applications. We already solved the first problem by being scheduler agnostic, but an I/O model was missing.

In this chapter we define a new I/O aware platform simulation approach, based on Batsim and his underneath simulation framework Simgrid. We expose a new Big Data applications model generated from Spark application traces, and a generic I/O transfer model capable to reflect the behavior of centralized and distributed file systems.

Experimental results show that thanks to Batsim, we can reproduce BeBiDa scheduling scheme and Big Data application performance metrics. We are also able to study file system trade-offs between centralized and distributed file systems: we find out that both file systems are equivalent for a simple application but the network access link to the centralized file system becomes an important point of contention when confronted to a larger data-intensive workload.

## Chapter's Contents

## 5.1 Introduction

Current HPC Systems are huge: they are composed of a lot of servers, from 100 to 100 000 nodes. The data movement also called I/O, is more and more the performance bottleneck, because of the new data-intensive usage of the platform and the fact that the file system is very costly [Use+10]. In the previous chapter, we have proposed a new approach to make Big Data and HPC applications run on the same HPC platform. Experiments on real platforms are promising but a lot of parameters are fixed and the size of the experimental platform is not as big as real platforms. In particular, we would like to study the I/O aspect more deeply, especially the trade-off between the use of a traditional HPC file system or a Big Data one. Studying HPC file systems in real conditions is not possible at scale because the energy/time cost of experiments is not affordable. For example, the BeBiDa experiments were carried out on 37 machines and it takes 4 hours to add one data point on the graph, omitting the data generation and ingestion. The problem that we are facing is not only the scale of the experiment but also the number of parameters to consider:

- Workload: application type and users' behavior.
- File system type: PFS or DFS or any hybrid FS.
- Storage bandwidth: capacity and number of disks.
- Network bandwidth: 1G, 10G, 20G, ...
- Scheduling policies: Easy backfilling, Conservative backfilling, ...
- The number of nodes.
- Network topology.

To be able to continue our study on the BeBiDa system, and more generally on the HPC and Big Data fields, we need a reliable simulator. In this chapter, we propose an I/O aware simulation approach in order to simulate HPC systems in the context of Big Data and HPC collocation. This study focuses on the replication of the BeBiDa experiment that was carried out on Grid'5000 (see Section 4.5), and try to extend the experiment parameters to explore the file system issues described above.

However, as an HPC platform is complex, modeling this whole system is difficult. Moreover, we need a model that is not too complex to avoid long computation and fastidious calibration, but precise enough to allow to study our problem. Batsim [Dut+16] is our approach to simulate an HPC and Big Data infrastructure. The platform model of Batsim is provided by the underneath simulation framework Simgrid [Cas+14], which has a large open source community and gathers a lot of efforts from scientists around the world to provide a fast and reliable computing infrastructure simulation. The platform simulation model is described in details in Section 5.3.

Thanks to Simgrid models, Batsim users can select the level of realism they desire to model the applications that compose the workloads: from delay jobs to very realistic one (SMPI [Deg+17]), which replays each message of an application along with the amount of computation in flops.

For this study, we have augmented the features of Batsim by implementing an I/O model that support both HPC and Big Data file systems described in Section 5.4. We also implemented a Big Data application model that relies on this I/O model, which is detailed in Section 5.5. The HPC application model is presented in Section 5.6.

Then, we have leveraged the Batsim network protocol to recreate the BeBiDa scheduling scheme with a message broker. This approach, which reproduces RJMS collaboration of BeBiDa on top of Batsim, is described in Section 5.7.

Finally, the multiple experiments that are conducted with these models are presented in Section 5.8. It is followed by a discussion on the possibilities offered by the new capabilities of the simulator and their limitations in Section 5.9. Finally, Section 5.10 concludes this chapter and gives future works.

## 5.2 Related works

Although multiple simulation approaches are similar to Batsim, they provide neither I/O model, nor multi-scheduler capacity [KTP16; GKN18; PNM09].

The Flux RJMS has been enhanced with I/O aware scheduling and avoids contention on the PFS using application's bandwidth requests [Her+16]. This approach evaluation is not based on simulation but on an emulator located inside Flux. They propose an analytic I/O hierarchy and contention model but it is limited to only two scheduling policies of this particular RJMS.

Firmament is a cluster scheduling algorithm for efficient job placement [Gog+16]. The evaluation of their algorithm is partly based on a simulator that replays a Google workload trace. The simulator intercepts remote procedure calls of the Firmament scheduler to make it run on top of a simulated infrastructure. This approach is similar to what Batsim does, but dedicated to Firmament only. A port of Firmament on top of Batsim is in progress.

Wrench [Cas+18] is very similar to Batsim because it is also based on Simgrid. The main difference is that Wrench is designed for performance evaluation of scientific workflow management systems, whereas Batsim is designed to experiment on parallel job scheduling and infrastructure modeling (network, storage, I/O, energy).

On the Big Data side, YarnSim [Liu+15] is a simulator of Hadoop Yarn based on the CODES/ROSS simulator. Internal mechanisms of YARN is simulated with logical processes that talk to each other, similarly to Batsim but with a more complex interaction that mimics Yarn's behavior. Only MapReduce application model seems to be supported and the code is not available, which prevents us to use it.

Based on the same simulation framework, Liu et al. studied the role of Burst Buffers on large HPC systems using 3 HPC applications traces on a specific Blue Gene/P infrastructure [Liu+12]. This simulator is tied to a specific platform and a very simple workload of only 3 applications, while Batsim, using a more coarse grain model, is able to run large workloads of 100s to 1000s of jobs on any platform.

To the best of our knowledge, our approach is the only computer infrastructure simulator capable to model multiple file systems with I/O contention on a platform and to simulate mixed workloads composed of Big Data and HPC applications.

## 5.3 Platform model

The platform simulation, including a network topology model and a network contention model, is provided by the distributed platform simulation framework Simgrid [Cas+14].

Simgrid is based on an Actor model, where each actor is located on one host of the platform and can communicate with the other actors through the network. Its network model is based on a fluid model that reproduces a TCP-like congestion model. It is a Max-min based model augmented with reverse-traffic support [Vel+13]. It allows to efficiently adapt the network traffic pace depending on the saturated link. However, as explained in this paper, this flow model has some limitation: if the network is under very high contention, in real execution some flow may completely collapse, i.e. do not transmit for a long period. The flow-level is not capable to capture this behavior. This is not considered as an issue because this kind of network collapse occurs in extreme cases, that can be avoided by sound application design.

Even if Simgrid provided flow-level model is more limited than the packet-level model, it gives similar results with much less computation [FC07].

On top of this actor and network flow model, Simgrid proposes two application models that are used by Batsim:

The SMPI model is described in details in [Deg+17]. This model is a way to accurately simulate MPI based applications very precisely: it spawns one actor for each MPI rank and replays all the MPI calls and the computation between calls. This preciseness comes with a cost in execution time which is affordable for application-level simulation but may be a limitation at the scale of an entire HPC platform. It is necessary to calibrate each platform in order to obtain accurate results, which can be a tedious time-consuming process that requires to have access to the actual platform targeted by the simulation.

The second model is the `parallel task` model. A parallel task takes in input a total amount of data to transmit between each host involved in the application, and the amount of computation that each node needs to achieve. It is very similar to the flow network model described before, but it slightly modifies it to be able to include the computation inside the constraint resolution. Thus, the progress of communication flows and computation of one application is coupled in the simulation on one progression value that depends on the current resources bottleneck, whether it is a network link or a computing unit. It implies that all computations and communications of one application are progressing at the same pace proportionally to their respective load. Hence, all the parallel tasks that are inside the application finish at the exact same time. It means that, depending on which resource is the bottleneck, only the resources that are limiting the performance are fully used, the usage of other resources are scaled down accordingly. This behavior is dynamic over time, i.e. updated at each event, and the resources are equally shared between the communication flows on the network links and between the applications on the computation unit.

Figure 5.1 illustrates the following example, considering a simple platform with two nodes with 1 CPU joined by one symmetrical network link.

First (1), the `Purple` (hatched) application is launched on the platform. It saturates the link but not the computing units because the quantity of bytes that have to be transferred regarding the capacity of the link is higher than the quantity of Flop regarding the CPU capacity.

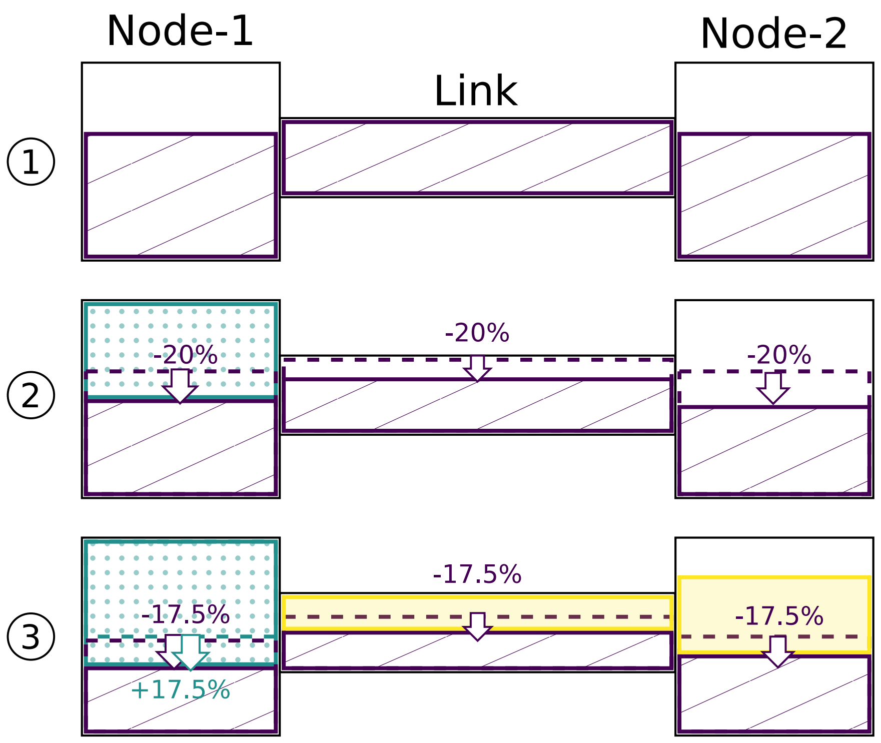


Figure 5.1.: Resource sharing example of 3 applications using the parallel task model running in concurrency: (1) `Purple` (hatched) application saturates the link but not the computing units. (2) `Green` (dotted) application takes half of the resources of node-1 and consequently reduce the computation power allocated to the `Purple`. (3) `Yellow` takes half of the link bandwidth.

Then (2), the `Green` (dotted) application takes half of the resources of node-1 and consequently reduces the computation power allocated to the `Purple` application by 20%, which makes node-1 the new performance bottleneck of `Green`. `Purple` directly adapts its progress to this new situation and reduces by 20% the resource consumption on other resources.

Finally (3), a third `Yellow` application is now started and takes half of the link bandwidth. This makes `Purple` change its bottleneck and reduce again its consumption on each resource by 17.5%. `Yellow`'s arrival also impacts `Green`, that can now take the extra computing power that was released by `Purple`.

This model is coarse-grained, but it is interesting for data-centric application modeling as it reflects the fact that if a task is waiting for data, this task will not fully use the computing resource; and in the other hand if the computation unit is saturated the application will reduce its data fetching pace.

## 5.4 File System Model

There are different kind of file systems. Centralized file systems rely on parallel access to disks and will be denoted by Parallel File System (PFS) — they are common in HPC platforms. Decentralized file systems will be denoted by Distributed File System (DFS) and are used in Big Data platforms. More detailed definitions are provided in Section 2.5.2.

There is a lot of variability in the implementation of those file systems: some PFS are hierarchical, compute nodes may not be connected directly to the centralized file system but go by intermediary nodes called I/O nodes. Another specificity of the PFS is the use of fast memory storage to buffer large I/O bursts called the `Burst Buffers` [Liu+12]. On the other hand, DFS are distributed on the compute nodes' disks to serve data as close as possible to the computation units. DFS topology is more simple, but they may be impacted by a large variety of storage units. A node can have multiple disks, and each of them can have different performance properties (NVM, SSD, HDD).

In order to explore a wide field of possibilities among these file systems in simulation, and even try to compose hybrid file systems, we need a flexible model.

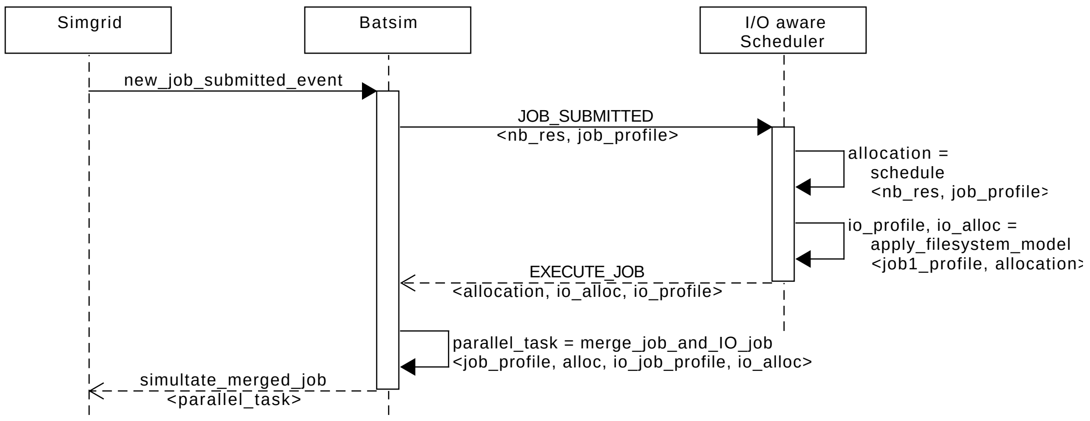
The model chosen for the storage is to use a simple Simgrid `Host`, with no computation power. The storage access latency and read/write bandwidth are modeled by two Simgrid `Links` that attach the storage node to the rest of the platform. This model enables to simulate any kind of file system topology, from classical DFS (by adding a storage node to every node of the platform) to centralized storage with a single large storage node, or any mix of the two.

Now that the platform contains storage node, we need to model the I/O transfer between them. At the time of the redaction of this chapter, Simgrid propose a storage model but does not feature a way to merge I/O traffic with a job's internal communications, which prevents us to use this model.

The Batsim implementation of the I/O transfers is based on the parallel task model described above. Every storage gets a resource ID that identifies it within the platform with a unique number, just like the computation resources. Using the storage ID and the normal resource ID, data movement is simply modeled by a matrix of communication involving computation node and storage nodes. The genericity of the model is ensured by keeping it outside of Batsim. Thus, the burden of maintaining the file system model is delegated to the user's defined process that is connected to Batsim, called the decision process or the scheduler.

The scheduler uses dynamic jobs to model the file system internal data movement like DFS load balancing, or data staging from PFS to Burst Buffers, but most of I/O transfers are triggered by applications. However, in order to simulate different file system configurations on the same workload, the application model needs to be independent of the file system. This way, the application model needs to contain enough information for the scheduler to generate actual I/O transfers at execution time. Thus, the application profile can be statically defined in the Batsim workload file, but the I/O traffic that is triggered by this application is dynamically generated by the scheduler depending on the context.

Concretely, when a job needs to be submitted, Batsim informs the scheduler of this event and attaches a job profile to this event. This job profile contains information about the quantity of I/O that needs to be done by the job, or any arbitrary information required by the file system model. Then, the scheduler can choose where the application will be allocated, and also, what are the I/O transfers that will occur during the execution of this application. Once the decision is taken, the scheduler sends the job allocation to Batsim inside the job execution order as usual but also includes an additional I/O job. This I/O job has its own allocation with the associated communication matrix that represents the transfer between the compute nodes of the jobs and the different storages. Finally, Batsim will merge this I/O job profile with the normal job profile before delegating the actual simulation to Simgrid.



Simgrid
Batsim
I/O aware Scheduler
new_job_submitted_event
JOB_SUBMITTED <nb_res, job_profile>
allocation = schedule <nb_res, job_profile>
io_profile, io_alloc = apply_filesystem_model <job1_profile, allocation>
EXECUTE_JOB <allocation, io_alloc, io_profile>
parallel_task = merge_job_and_IO_job <job_profile, alloc, io_job_profile, io_alloc>
simultate_merged_job <parallel_task>

Figure 5.2.: This sequence diagram illustrates how the Batsim file system model is implemented. When a new job submission event occurs in the simulation, Batsim informs the scheduler, which chooses the job's allocation and generates an additional I/O job. This I/O job is then merged with the actual job by Batsim in a single parallel task that is then sent to Simgrid for simulation.

Here is an example of this mechanism illustrated by a sequence diagram in Figure 5.2. Let's assume that `job1` was submitted to the scheduler with a request of 2 computing resources. The profile of the job is defined by a Json object, `job1_profile`. The following illustrates this example.

1. The scheduler receives the job submission from Batsim with this forwarded profile, where the quantity of I/O is just information given to the scheduler (not read by Batsim):

```
"job1_profile": {
    "type": "parallel_homogeneous_total",
    "cpu": 1e10,
    "com": 1e7,
    "io_reads":  8e8,
    "io_writes": 2e8
}
```

2. The scheduler takes the decision to allocate this job on the resources 0 and 1. Then it decides, regarding its file system model, that each node will read half

of the data from only one centralized storage (located at the resource id 10), but the nodes of the application will write on their local disks (id 2 and 3). A new profile of type `parallel` is generated and submitted to Batsim with the name `io_for_this_alloc_on_job1`. The kind of job profile is composed of a computation matrix that may reflect I/O related computation (here set to 0), and a communication matrix that represents the aforementioned assumptions. Note that the direction of the transfer is from row to column, and that the indices of the matrix are mapped to the I/O allocation, e.g. for the allocation of $(0, 1, 2, 3, 10)$ a value of $4e8$ on the 1st column and at the 5th row means that 400 MB will be transferred from the resource 10 to the resource 0.

```
"io_for_this_alloc_on_job1": {
    "type": "parallel",
    "cpu": [0   ,0   ,0   ,0   ,0]
    "com": [0   ,0   ,1e8,0   ,0
            0   ,0   ,0   ,1e8,0
            0   ,0   ,0   ,0   ,0
            0   ,0   ,0   ,0   ,0
            4e8,4e8,0   ,0   ,0]
}
```

3. The scheduler asks Batsim to execute the job with the given job allocation, and the additional I/O profile.

```
{
    "job_id":"08a582!1",
    "alloc":"0-1",
    "additional_io_job": {
        "alloc":"0-3,10",
        "profile": "io_for_this_alloc_on_job1"
    }
}
```

4. Batsim merges the 2 profiles, and generates a new profile with the I/O matrix that is then sent to Simgrid to be simulated:

```
{
    "alloc": "0-3,10",
        "cpu": [5e9,5e9,0  ,0  ,0],
        "com": [0  ,5e6,1e8,0  ,0
                5e6,0  ,0  ,1e8,0
                0  ,0  ,0  ,0  ,0
                0  ,0  ,0  ,0  ,0
                4e8,4e8,0  ,0  ,0]
    }
```

Note the difference of allocation between the job itself and the I/O that it generates. Batsim is capable to merge any interval set of resource allocation, even if only part of the job's nodes are taking part in the I/O transfer.

This simple file system model is generic enough to simulate any centralized and decentralized file system because it does not assume any kind of I/O behavior. For example, it is possible to simulate hierarchical file system like a PFS with I/O nodes, or a multi-tiers storage setup with two different centralized file systems, e.g. NFS and Lustre, or even a mix of DFS and PFS.

It is fully dynamic: the I/O transfers inside the application are generated online by the decision process, which allows taking the I/O into account for any decision, from job allocation to I/O gateway selection. Also, dynamic jobs can be created to model internal file system I/O transfers at any time during the simulation.

This model also have limitations, that are discussed in the Section 5.9.

## 5.5 Big Data Application Model

In order to simulate converged platforms, we need an application model that reflects the Big Data workload. This section describes our approach to model Big Data applications. This model is complementary to the file system model defined in the previous section.

### 5.5.1 Model requirements

To be able to model Big Data applications properly, we need to take into account the main properties of this kind of data-centric workload. Also, some properties are necessary to enable the study of converged platforms managed by BeBiDa. The following enumerates these properties:

1. Malleability: a job is able to dynamically adapt to the modification of the number of resources that are allocated to it.

2. Preemption and failure support with a cost: the preemption of parts of the resources has an effect on the job execution that depends on when the preemption happens.

3. File system agnostic I/O: to be able to compare the different file system options the application model needs to work on both PFS, DFS, or any kind of hybrid file system.

4. I/O bottleneck effect: we want the simulation to reflect the effect of I/O bottleneck that may slow down the applications.

5. Time effective: the time to simulate a large platform over a long period needs to be reasonable. This has an impact on the model accuracy, but, as we simulate at a more coarse grain platform level, we can tolerate model imprecision at the application level.

In order to define a model that is consistent with reality and fulfill these properties, we need to understand how the main Big Data frameworks are working.

We choose to look at the Spark [Fon19c] framework which is very popular in academic and industrial communities.

### 5.5.2 White Box Model Based on Spark Traces

The first two properties of the requirements expressed above (about malleability and preemption support) are closely related to the Big Data application runtimes, typically MapReduce (MR) and Spark. The schema in Figure 5.3 shows the MR and Spark applications internals.

As described in Section 5.3, Simgrid is based on a fluid model, and if an entire MR application is modeled with only one Batsim job profile, the resource usage would be flattened for the whole job duration. Thus, it would not reflect the different

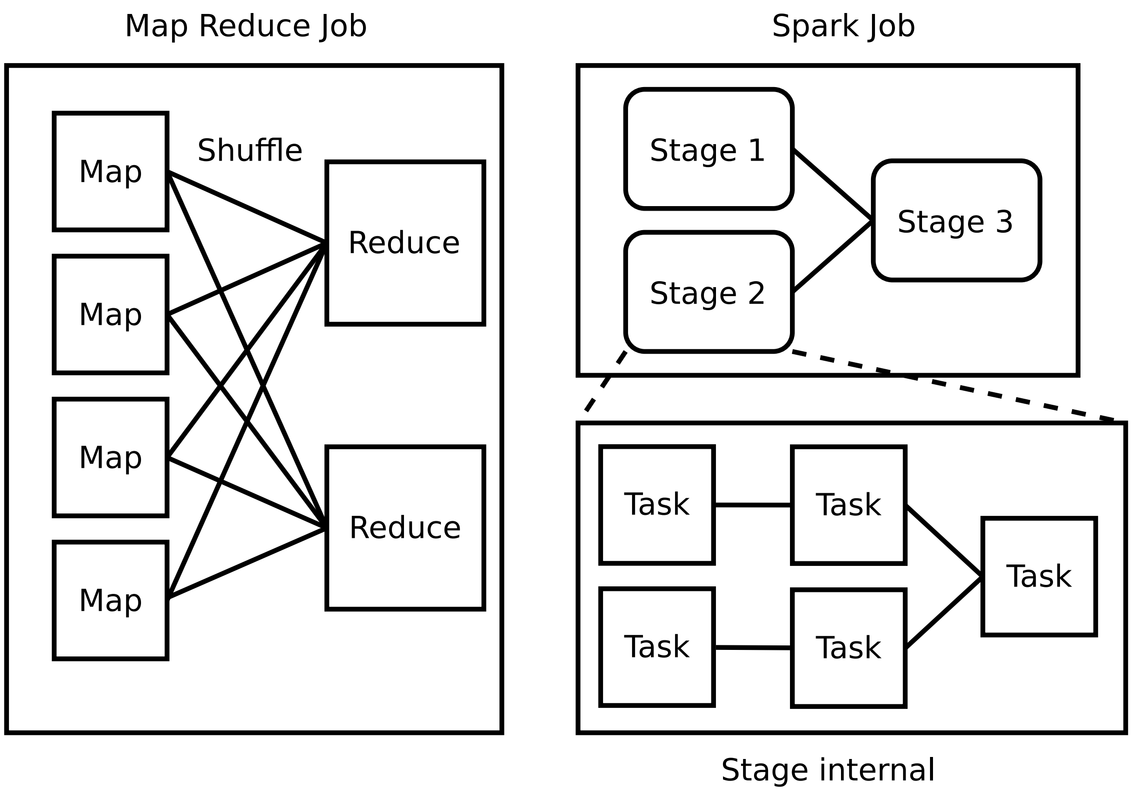


Figure 5.3.: MapReduce and Spark internal structures: a MapReduce application is always composed of 3 phases: Map/Shuffle/Reduce, whereas Spark applications are split into a DAG of Stages themselves split into a DAG of tasks.

phases properly. So, each phase can be modeled in a separate job profile, and then put in a sequence profile, i.e. a sequential list of jobs to be executed in order.

However, MapReduce is now supplanted by Spark, whose model is more effective as it takes a large allocation at the beginning of the application and then starts a process called executor on each container. The application is then split into tasks scheduled on the executors by Spark itself. In contrast to MapReduce, this mechanism has the benefit of keeping every intermediate result in-memory, thus greatly decreasing the application execution time. Even if Spark is more complex than MR, it is based on the same principles that consist of splitting the applications into phases called stages. A Spark stage is defined by a set of tasks with narrow dependencies, i.e. each data block is processed sequentially on an executor without inter-executor synchronization. As in the MapReduce paradigm, this synchronization phase called “shuffle” takes place between two stages.

Hence, the model of sequence profiles defined above can also be applied to Spark at the stage level, i.e. all stages are described by job profiles, that are put in sequence to form the entire Spark application profile. Note that Spark stages can have more complex dependencies than a sequence based on a directed acyclic graph, but, as it seems not very common regarding the benchmarks we have used, this is left for future works.

This representation permits the simulation to have a realistic effect on resource preemption (or failure). Indeed, the shuffle phase that separates each stage acts as a checkpoint for the intermediate results of the application, so that a stage can be considered as atomic. It means that interruption leads to re-computation, but only for the current stage and not for the previous ones.

This implies to use a “white box” model that generates these profiles by extracting metrics from Spark application logs. Fortunately, Spark provides detailed traces for each application that contains the total amount of I/O, CPU, and Memory consumed by each task. These metrics need to be aggregated by stages in order to get coarse grain simulation that meets the time effectiveness constraint.

For this purpose, we have implemented a generator script that takes a set of application traces in inputs, as provided by the standard Spark History Server, and generates a set of application profiles[1].

[1]This script is available here: `https://gitlab.inria.fr/batsim/bebida-on-batsim/blob/master/experiments/BDA_workload_generation/spark_result_to_batsim_csv.py`

### 5.5.3 Implementation in Batsim

In order to allow a job's malleability, the job's allocation needs to be dynamically resized, and the remaining work needs to be spread out on the new resource set. Batsim does not provide this kind of feature: jobs have static allocations and once a job is started it is not possible to change the amount of work that needs to be done on each resource. This is due to Simgrid limitation, but also because Batsim does not assume any application model in order to stay generic and simple.

But it exists a way to achieve this in Batsim: when the number of resources attributed to a job changes, the job is killed, and then re-submitted on the new set of resources with an equal amount of work allocated to each resource.

To do so, we have implemented two features in Batsim. First, Batsim needs to expose the quantity of work already done (before killing) to be able to reflect the cost of preemption correctly. The second feature is a new kind of job profile that takes an amount of work to spread evenly on the allocated resources at runtime, thus being independent of the number of resources.

Based on these features, the rest of the model is implemented in the external decision process that plays the role of the scheduler but also manages the file system.

### 5.5.4 Implementation in the scheduler

Using the progress ratio now exposed by Batsim and the new job profile type, the scheduler is now capable to reflect the preemption of resources cost. When all or part of the resources allocated to an application is preempted, the current stage is considered lost and the application is restarted from the beginning. Note that in the case where only part of the resources of a job are preempted, the totality of the work of the current stage must be redone. This is slightly different from the real behavior of Spark which is more complex, but because of the cost of rescheduling and the data movement that may be triggered, this is considered a close enough approximation.

To meet the requirement to be file system agnostic, the I/O model is also owned by the decision process attached to Batsim, in this case, the Big Data scheduler defined in Section 5.7.2.

A simple file system model is implemented in the scheduler: for each I/O phase, it randomly distributes data blocks that will be read on the cluster's nodes with a control over locality. The percentage of locally accessed blocks is picked randomly with a uniform law between 60% and 80% for each job at execution time. This distribution was chosen according to empirical data on the locality distribution obtained from the Bebida real conditions experiments (See Section 4.5 for more details). The block write operations are done similarly to the HDFS implementation: one local replica, and two remote copies that are located on two different (and randomly selected) nodes on the cluster.

This model has the advantage to give us the control of data locality, which is interesting for what-if experimentation. It also gives the ability for the scheduling algorithm to remain simple because it does not have to be locality-aware.

Another option is to place data blocks randomly at the beginning of the simulation and to keep a block location index just like a file system metadata server would. So, depending on the current state of the file system, the scheduler consults its index to find the wanted blocks and then allocates resources to the job with the best data locality available. This more realistic model was implemented by the Firmament simulator [Gog+16]. But, because of the cost of implementation of this model and the loss of control over data locality that comes with it, this fine grain model is left for future works.

## 5.6 HPC Application Model

Like the Big Data application model, HPC application model needs to be coarse-grained, because we are simulating at the platform level with large clusters and potentially hundreds of simultaneous jobs.

This model also needs to expose the I/O behavior of the applications in order to be able to study contention effects on the different file systems.

Simgrid is already supporting an HPC model based on SMPI (described in Section 5.3). But this model does not meet the constraints previously defined: it is too fine grain to be able to simulate a complete cluster workload on a reasonable time, and above this, it does not feature any I/O model.

We choose to use the simplest model that Batsim propose which is the “delay” model that is just a time to wait without any activity simulated on the platform.

We planned to create a new model based on the same level of model as the Big Data application model using periodic I/O phases to split HPC application in sequences of profiles, similarly to the one proposed by Herbein et al. [Her+16].

The HPC modeling issues are discussed more thoroughly in Section 5.9.

## 5.7 Reproduce BeBiDa scheduling scheme

The way the BeBiDa collaboration system makes Big Data and HPC schedulers work together is based on HPC jobs’ prolog and epilog. This mechanism is described in details in 4.4. It requires that each HPC job starting and completion event triggers an action on the Big Data RJMS to remove or add resources from its control. We need to find a way to reproduce this in simulation.

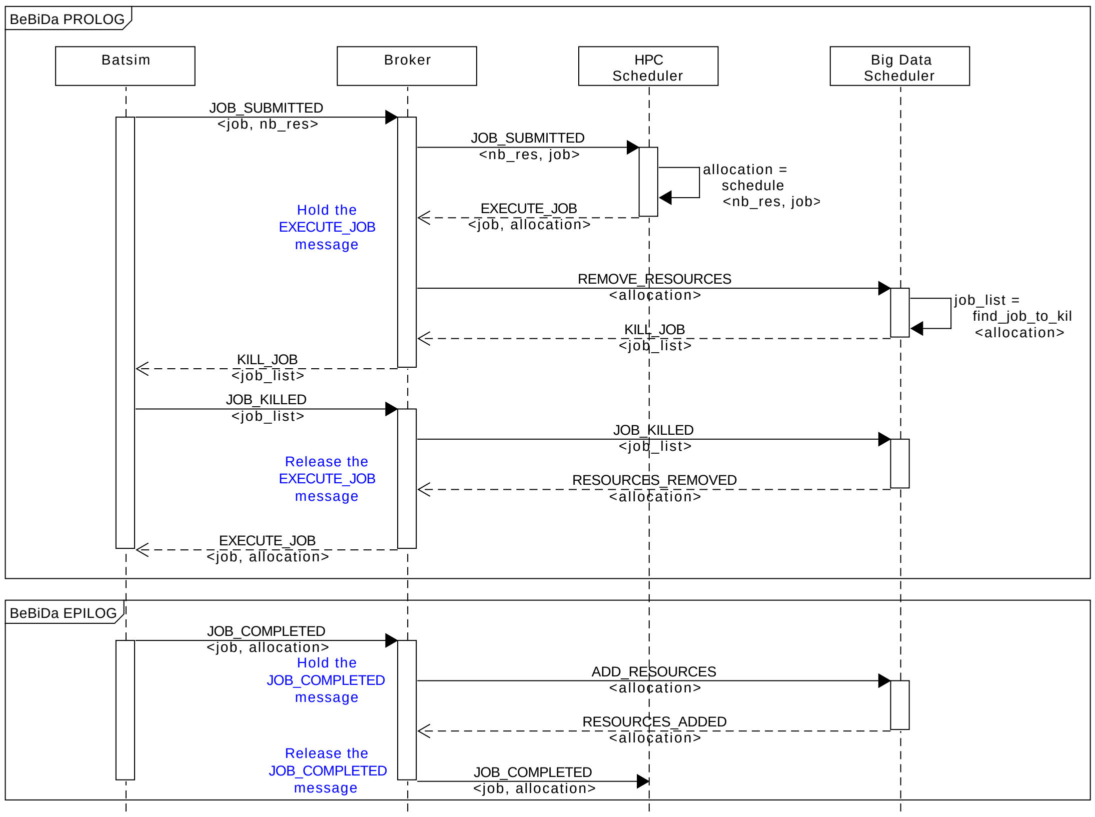
Another issue is that Batsim has only one endpoint to connect to a scheduler, but we need to connect two schedulers to it, each of them managing its own workload. Batsim is capable to manage multiple workloads, that are uniquely identified, but it does not support multiple schedulers to avoid complex time synchronization.

In order to implement the BeBiDa scheduling scheme on top of Batsim which addresses those issues, we chose to add a layer between Batsim and the two schedulers. This layer is called the broker because it works as an intermediary between Batsim and the schedulers.

### 5.7.1 The Broker

The broker enables multiple schedulers to talk to Batsim while implementing the BeBiDa prolog/epilog mechanism. To do so, it routes the messages from Batsim to the different schedulers and implements the resource sharing logic by sending and receiving additional messages. Then, it merges the schedulers response and send it back to Batsim. The sequence diagram in Figure 5.4 illustrates this mechanism.

In details, the broker receives the Batsim messages, and regarding the workload, routes the message to the appropriate scheduler. Then, it receives the responses



BeBiDa PROLOG
Batsim
Broker
HPC Scheduler
Big Data Scheduler
JOB_SUBMITTED <job, nb_res>
JOB_SUBMITTED <nb_res, job>
allocation = schedule <nb_res, job>
Hold the EXECUTE_JOB message
EXECUTE_JOB <job, allocation>
REMOVE_RESOURCES <allocation>
job_list = find_job_to_kil <allocation>
KILL_JOB <job_list>
KILL_JOB <job_list>
JOB_KILLED <job_list>
JOB_KILLED <job_list>
Release the EXECUTE_JOB message
RESOURCES_REMOVED <allocation>
EXECUTE_JOB <job, allocation>
BeBiDa EPILOG
JOB_COMPLETED <job, allocation>
Hold the JOB_COMPLETED message
ADD_RESOURCES <allocation>
RESOURCES_ADDED <allocation>
Release the JOB_COMPLETED message
JOB_COMPLETED <job, allocation>

Figure 5.4.: This sequence diagram illustrates the BeBiDa prolog/epilog mechanism implementation by the Broker. Both are based on the same technique: the Broker is holding a message transferred between Batsim and the HPC scheduler, while it asks the Big Data scheduler to add or remove resources. When the resources status is updated by the Big Data scheduler, the message is released.

from the schedulers, merges them, and replies to Batsim. In order to apply the resource sharing logic, it is also able to intercept certain messages. The following example depicts the BeBiDa prolog/epilog implementation inside the broker.

When a job has to be started by the HPC scheduler, it sends a message that contains the execution order along with the job allocation. The broker receives this message and before sending it to Batsim, it intercepts the message. Then, it asks the Big Data scheduler to decommission the nodes present in the job's allocation. When the Big Data scheduler receives this order it asks Batsim to kill the jobs that were running on those resources and probably resubmit a new job in order to complete the work that left from the killed jobs. Only then, the HPC scheduler's execution message will be sent to Batsim. This is the HPC job prolog.

The epilog is more simple: when an HPC job finishes, the broker recommissions the associated nodes to the Big Data resource pool before telling the HPC scheduler that the job is over.

The broker source code is roughly 400 lines of Go, and it is publicly available[2].

### 5.7.2 The Big Data scheduler

This Section contains a description of the Big Data decision process, which also implements the distributed file system model, and manages the application resource scaling. Most of its behavior have already been in the application model Section (5.5.4) above. Hence, this section focuses on the scheduling part of this decision process.

The scheduling policy applied to the Big Data applications is `First Come First Served` (FCFS). This is the default policy applied by the scheduler of YARN, the Big Data RJMS use in the BeBiDa real conditions experiments.

The set of resources available to the scheduler is dynamic: when some resources are preempted, or released, by the HPC RJMS, the Big Data scheduler received a remove, or add, resources order respectively. Note that these two messages are the only additional logic added to the scheduler which is not present in the Batsim message protocol. So, when one of this event happens, the scheduler load balance the application to enforce the FCFS policy regarding the resource requests of each

[2]https://gitlab.inria.fr/batsim/batbroker

application. Once a set of resources becomes available, the first application on the queue is always granted by as many resources as possible until it reaches the total amount of resources requested at submission time. Then, the same policy applies to the next application in the queue, and so on.

The implementation of this mechanism is done using a `kill and resubmit` method: every time an application resource allocation change, the application is killed by the scheduler, then a new job profile corresponding to what left of the job's work is created and the job execution will continue on the new set of resources. If the application was killed by a resource preemption, the cost of this preemption is applied by restarting the current application stage from its beginning.

This scheduler is open source and publicly available[3].

### 5.7.3 HPC Scheduler

The strength of this two-level approach is to totally decorrelate the HPC scheduling from the scheduler's coordination mechanism. It means that any HPC (or Big Data scheduler) can be used on top of the broker as long as it is compatible with the application profiles used in the workload.

However, for a non-trivial file system modeling, the file system model needs to be implemented in the scheduler, thus limiting the choice of HPC scheduler that can be used. No HPC scheduler compatible with Batsim is implementing a file system model for now, this is discussed in the Section 5.9.

## 5.8 Experiments

This section presents the experiments done and envisioned using the I/O simulation capabilities of Batsim.

The first milestone in the development of the I/O aware simulation is to be able to explore the possibilities offered by the BeBiDa approach, and also find its limitations. Using the Big Data model generated from the Spark traces, and a simple delay for the HPC application model, we are able to reproduce the results of

[3]`https://gitlab.inria.fr/batsim/pybatsim/blob/master/schedulers/schedBebida.py`

the real condition experiments in simulation and to extend them with file system study.

### 5.8.1 Calibration

Before running the actual simulation, the Big Data application model needs to be calibrated in order to obtain simulated execution time close to the real one. To do so, we have run each application on the cluster that was used for the real conditions experiments then we have compared actual execution time with the simulated execution time.

The first finding of this calibration is that most of the simulated applications are running faster than in real conditions. The main reason is the fact that we have set the local disk bandwidth to a system level raw performance value whereas our model is based on application level values. That is why we have created a new storage profile called “low-HDD” that diminishes the read and the write bandwidth by a third in order to reflect the overhead of the file system.

The second observation is that in our model the applications are scaling linearly whereas in real conditions they are not. The weak scalability, i.e. the scalability with a fixed amount of work per resources, is depicted in Figure 5.5. This difference in the scaling behavior can be due to multiple factors. One of them is the fact that we are not taking into account scheduling overhead at the job level and at the task level. Also, the parallel task model flattens the resource usage which may hide some temporary bottlenecks inside the application stages. Moreover, the Simgrid flow model is optimistic in the case of network contention, which may lead to higher bandwidth usage in simulation.

Regarding the previous statement, we can define our application model as an upper bound of the application performances. Also, this calibration process is simple and can be extended to a complete model evaluation which is discussed in Section 5.9.2.

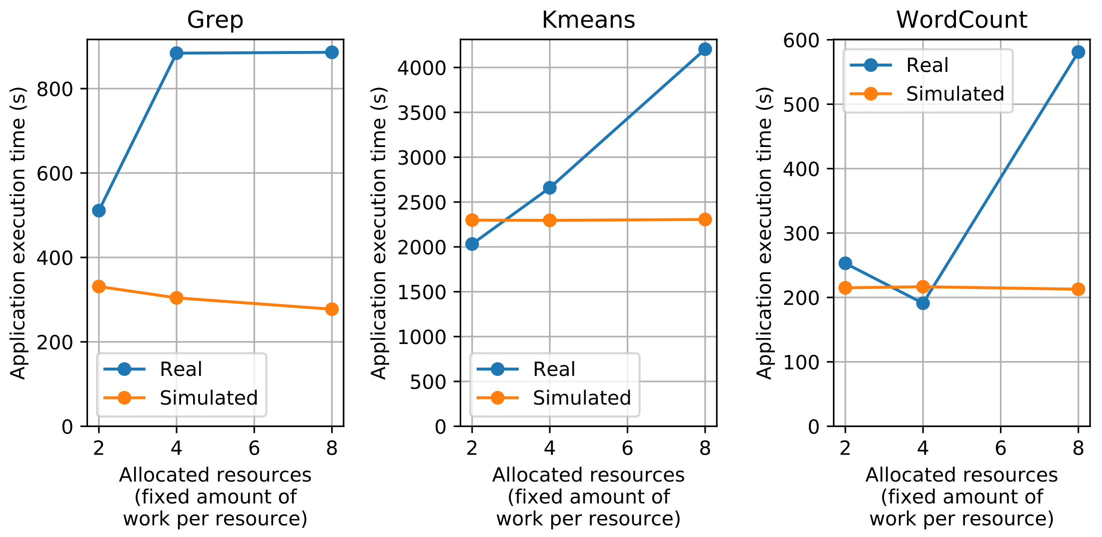


Figure 5.5.: Weak scalability of Spark application in simulation and in real conditions for 3 benchmarks. Simulation scales linearly but not real conditions experiments.

### 5.8.2 Reproducing BeBiDa real conditions experiments results

In order to reproduce BeBiDa real condition experiments in simulation, we need to feed the simulator with inputs that reproduce the BeBiDa experiment previous conditions:

- The HPC and the Big Data schedulers, with the prolog/epilog mechanism implemented by the broker, that are detailed above in Section 5.7.
- The platform that mimics the characteristic of the real platform Graphene (32 nodes, 1GB Ethernet, 4 cores), in the form of an XML Simgrid platform file. We also use for the file system study several platforms of the same size but with different disk capacity and file system, as generated in the experiment notebook[4]. As explained in the previous section, the storage bandwidth of the nodes was calibrated to get closer to the real execution time.
- A model of the workloads that were used for the Big Data and HPC part in the form of Json Batsim workload files. This format was conveniently already used for the real condition experiment, so only the job profiles part of the workload have been adapted.

[4]https://gitlab.inria.fr/batsim/bebida-on-batsim/blob/master/experiments/bebida-exp-replay/Bebida_on_batsim_simulation.ipynb

To be able to compare the experimental results of real and simulated conditions, we need to reproduce the results of the former in the latter. The first result of the simulation is that we are able to reproduce the BeBiDa scheduling scheme. The Figure 5.6 compares the real and the simulation execution of the same example workload. We can observe the differences between the two Gantt charts. The HPC scheduling is slightly different: this can be due to the fact that we are not using the exact same scheduler implementation and that in real conditions the scheduler adds a 1-minute guard time between each job. The Big Data jobs are shown as small bars that correspond to the Spark executors in real conditions, but in simulation, we see rectangles because of the application model. More importantly, we can observe that the execution time of the Big Data applications is underestimated by our simulation model. Indeed, there is unused space on the simulated cluster whereas in real conditions the cluster was full all the time.

The second results that we want to compare with the previous experiments is the efficiency of the Big Data applications. However, computing the exact same metrics is impossible, i.e. the Big Data Time Effectiveness, because this metric is based on the task execution time and the application model does not provide this level of detail.

However, computing a similar metrics is possible, but based on the total execution time of the application. To do so, we have run control simulation where each type of Big Data application run alone on the platform with a 100% data locality in order to get its execution time in the best possible conditions. Then, we have computed a time effectiveness metric as a ratio of the best execution time over the actual execution.

Figures 5.7 and 5.8 display a comparison of the time effectiveness between the real conditions experiment (on the left) and the simulation (on the right). Even if these graphs do not match completely, we can observe similar trends on the variability of this metrics regarding the different combination of workloads.

These results give us confidence on our model to evaluate BeBiDa on other parameters like the storage and the network bandwidth, but also the different kind of file systems.

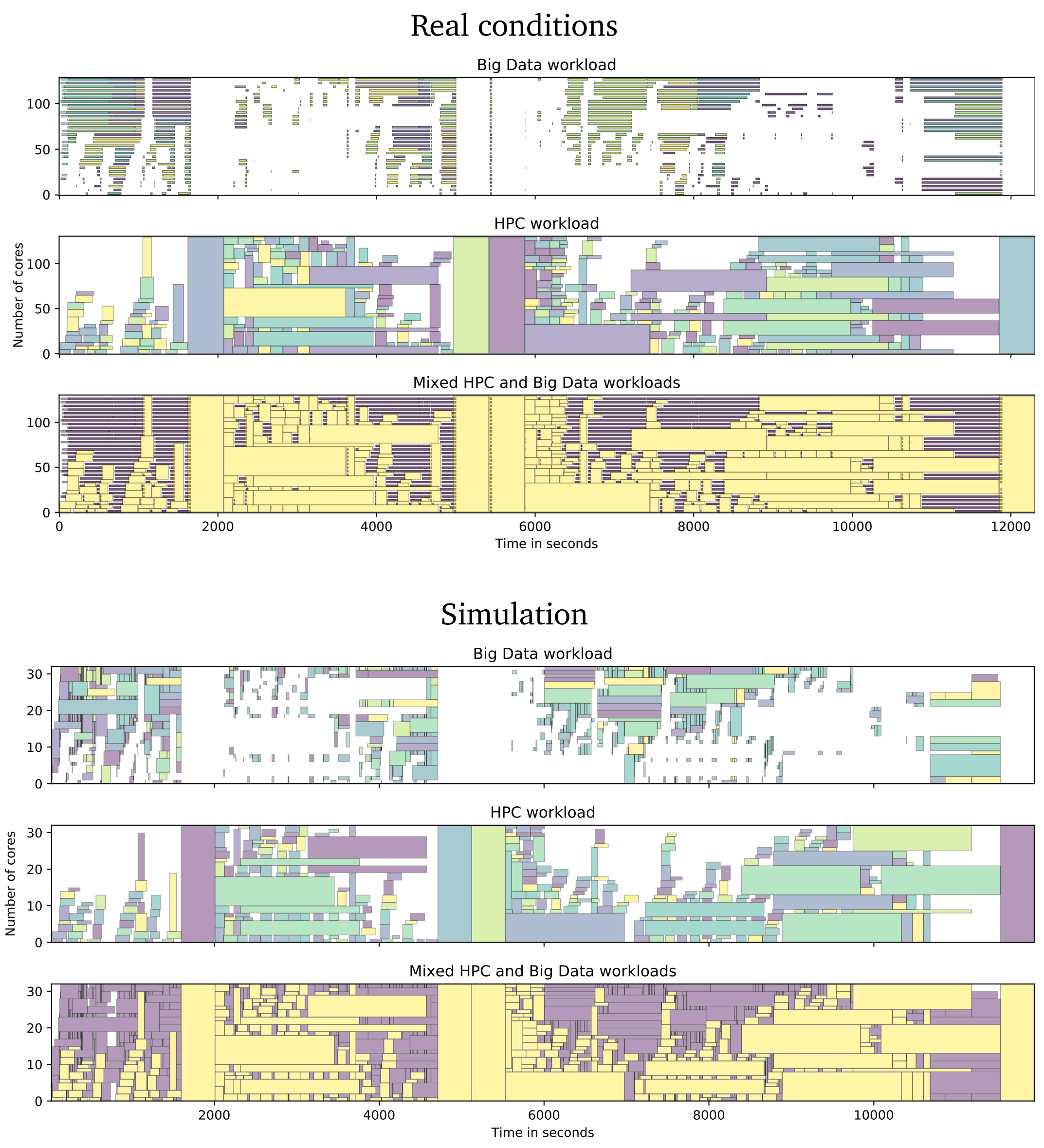
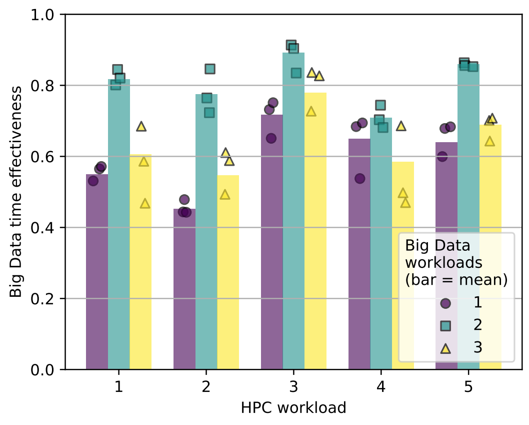


**Figure 5.6.:** Example of one experiment with BeBiDa in real condition (upper) and in simulation (lower).

### 5.8.3 Exploration of File System options

One of the strengths of the proposed application models is that they are generic enough to work on any file system. Thus, we can explore and compare the different options that can be selected for a platform file system for our hybrid workload.

We can study the trade-off between DFS and PFS (or a mix of the two), i.e. how much network and disk bandwidth we need on a PFS in order to get the same



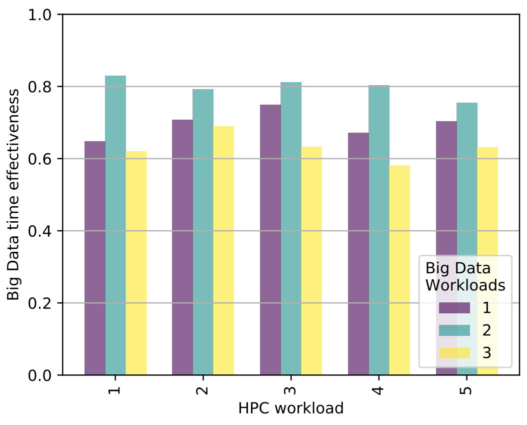


**Figure 5.7.:** Big Data Effectiveness in real conditions. Each point represents a one real execution instance, the bar is the mean.

**Figure 5.8.:** Big Data Effectiveness in simulation. The simulation is deterministic so the bars represent one simulation instance

efficiency as a DFS for data-intensive workloads. Several studies seem to indicate that DFS is more suitable for Big Data application than PFS (see Section 2.5.2).

We will try to compare the application performance regarding different types of file systems, types of disks, and different network capacities between the nodes and between the nodes and the PFS when this type of file system is used.

The two file system types are the parallel file system (PFS) and the distributed file system (DFS) defined more extensively in the Section 2.5.2.

The selected platform is the same as in the BeBiDa experiment: 1 master node and 32 compute nodes, all attached to a single switch. The network capacity is one of the experiment parameters that varies between 1 Gb, 10 Gb, and 20 Gb of symmetrical bandwidth.

The PFS type of storage is represented in simulation by a single node, attached to the rest of the cluster by special asymmetric network link that emulates the file system read and write bandwidth. This link is set in both directions to the minimum bandwidth between the network capacity of the link and the file system bandwidth regarding the traffic direction, i.e. read or write. This file system bandwidth is set to the cumulated bandwidth of 16 storage disk, which is a common setup for this cluster size, with different per disk bandwidth. We have selected 4 types of storage disk bandwidth, also called capacity, listed in Table 5.1. Note that because the Big Data workload is accessing data by large sequential blocks, only the sequential operations are considered. HDD and SSD sequential read and write

| Storage type | Sequential read | Sequential write |
|---|---|---|
| HDD-low | 100 MB/s | 60 MB/s |
| HDD | 150 MB/s | 80 MB/s |
| HDDx2 | 300 MB/s | 160 MB/s |
| SSD-low | 450 MB/s | 225 MB/s |
| SSD-high | 1450 MB/s | 1450 MB/s |

Table 5.1.: HDD and SSD sequential read and write was extracted from recent benchmarks results. HDD-low is the disk capacity with 30% diminished capacity due to file system overhead.

were extracted from recent benchmarks results[5][6]. HDD-low is the disk capacity with 30% diminished capacity due to file system overhead.

The same set of disk capacity parameters is used for the DFS file system capacity, which uses the disks attached to each compute node.

All the parameter combination along with control experiment represents 1920 simulation that runs in approximately 13 hours on a laptop. This experiment simulation and analysis are fully reproducible thanks to publicly available notebooks[7][8].

The results of these simulations show that depending on the file system used, the applications performance is not sensitive to the same parameter. The DFS is sensitive to the storage type, whereas the PFS is sensitive to the capacity of its network link to the cluster. This is expected because the performance of the DFS is highly dependent to the local storage bandwidth, and because the PFS main point of contention is the unique network link that attaches it to the cluster.

In Figure 5.9 we can observe the trend on the execution time distribution of the applications running alone. The reader can also note that a good performance gain is achieved by switching from one to two hard drives (`HDDx2`), and that it then starts to plateau. For the PFS, the same level of performance is reached with a 10G PFS network link, but there is no difference between the 10G and the 20G capacity.

[5] https://openbenchmarking.org/result/1103091-IV-1103089IV49
[6] https://openbenchmarking.org/result/1810208-SK-CORSAIR5106
[7] https://gitlab.inria.fr/batsim/bebida-on-batsim/blob/master/experiments/bebida-exp-replay/Bebida_on_batsim_simulation.ipynb
[8] https://gitlab.inria.fr/batsim/bebida-on-batsim/blob/master/experiments/bebida-exp-replay/Bebida_on_batsim_analysis.ipynb

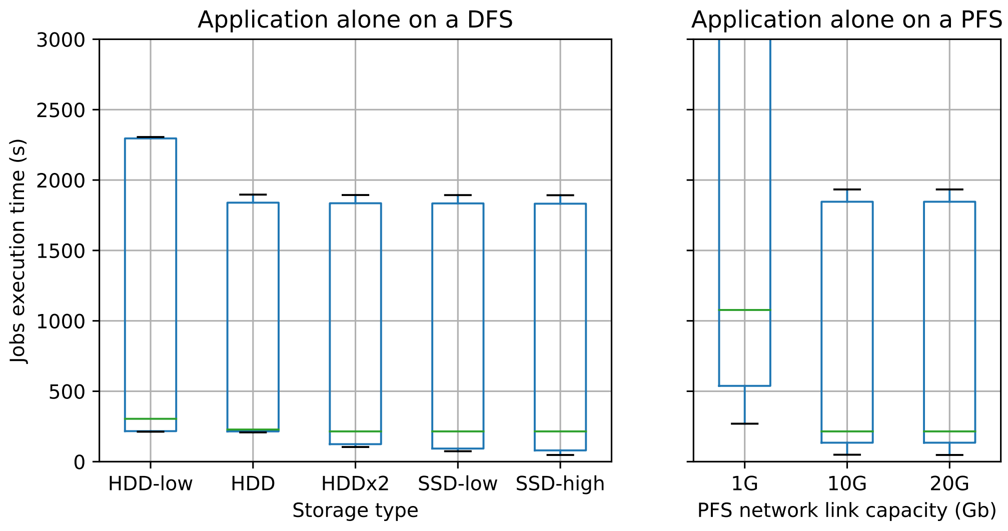


Figure 5.9.: Execution time of Big Data applications running alone on the cluster. Applications' performance using a DFS is sensitive to storage type, whereas it is sensitive to the capacity of the file system link to the cluster using a PFS. Note that for the DFS case, using a disk better than HDD only impact a small portion of the applications.

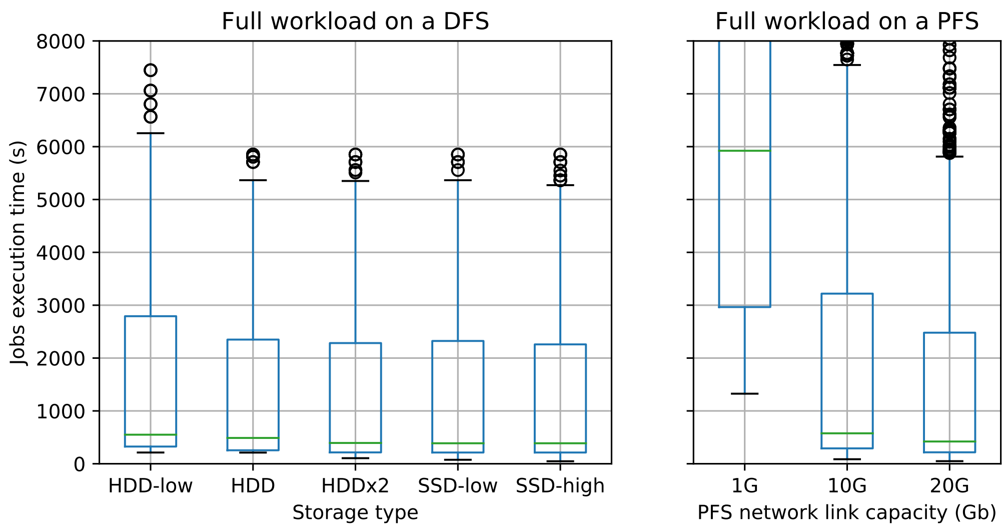


Figure 5.10.: Execution time of 3 different workloads of 25 Big Data jobs running on a 16-node cluster. Applications' performance using a DFS is sensitive to storage type, whereas it is sensitive to the capacity of the file system link to the cluster using a PFS. Note that for the DFS case, using a disk better than HDD only impact a small portion of the applications.

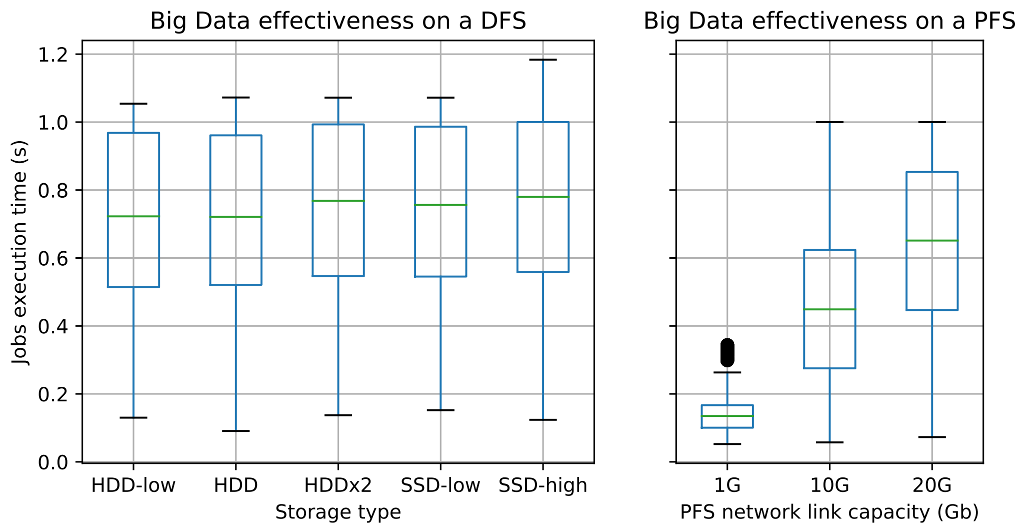


Figure 5.11.: Big Data time effectiveness (see 4.5.2) in a BeBiDa setup regarding different file system options. Applications' performance using a DFS is sensitive to storage type, whereas it is sensitive to the capacity of the file system link to the cluster using a PFS. We can observe that in the DFS case the impact of the disk capacity exists but it is small, while for the PFS case, the PFS network link capacity is highly impactful.

This last statement does not hold when a full workload of 25 Big Data jobs is submitted to a 16-node cluster. Figure 5.10 shows that the contention on the file system created by the application running in concurrence clearly harms the applications' performance using a PFS, while in presence of a DFS the applications' performance remains not impacted.

These results are corroborating experimental results carried out on Grid'5000 by Yildiz et al. that compares Spark applications performance on the two types of file systems [YI18].

These results show that, as expected, the DFS is more suitable to cope with a Big Data workload than the PFS because of the contention avoided by local data access. Interestingly, it also shows that the use of SSD on local nodes for a DFS seems to be unnecessary because having HDD on each node achieve the same performance even in the presence of a full workload.

The choice of the file system also impacts the Big Data time effectiveness (see 4.5.2) within a BeBiDa setup. The Figure 5.11 depict the fact that the Big Data applications are more effective in case of resource preemption when using a DFS file system. Note that some values are exceeding 1 in the DFS case: it can be explained by the fact that the control experiment with 100% data locality that is used as comparison is sometimes slower than the same application with a lower data locality. Indeed, reading every byte locally may saturates the local disk bandwidth, but if part of the data is access remotely it increases the data access parallelism and thus diminishes the data access time.

## 5.9 Discussion and future works

This section presents future works and works in progress because lots of projects were initiated around this new I/O aware capacities of Batsim, but most of them are not finished before the end of this thesis.

### 5.9.1 BeBiDa in-depth evaluation

Lots of aspects of the BeBiDa mechanism have not been evaluated yet. It would be interesting to implement the simulation of other solutions like multi-level scheduling (YARN [Vav+13a], Mesos [Hin+11], ...) or schedulers adaptor (Intel Hadoop HPC scheduler adaptor [Int17]) in order to compare them with our approach and to explore the different trade-offs.

We would like to extend experimental results presented in Section 5.8 by studying larger platforms in conjunction with more complex workloads with a wider type of application and resource requests. For example, what are the thresholds on the HPC workload utilization level from which the Big Data effectiveness collapses or plateaus? Furthermore, we would like to explore all the BeBiDa future works exposed in Section 4.6: avoid preemption with blind or explicit collaborations between the RJMS like improving Big Data quality of service creating holes in the HPC scheduling. Also, the Time fragmentation metric used as scheduling objective for the HPC scheduler. We also would like to study more in-depth the DFS replication and placement efficiency on data locality by testing different allocation policies.

On the PFS side, we can also study the different integration possibilities of the file system in the cluster topology, and find which scheduling policies are the most suitable depending on this choice.

We also want to study the use of our approach in the context of Cloud computing, where the workload is composed of long-running services in a virtualized environment using migration instead of resource preemption.

### 5.9.2 Big Data model evaluation

With Adrien Faure, we are working on an experimental evaluation process of the Big Data model along with the described in Section 5.5. This evaluation is an attempt to validate this model by comparing real Spark execution with simulation. This validation is twofold:

**Execution time conservation for different application traces:** First, we want to compare execution time in real conditions and in simulation for different application traces with different parameters (application type and input size). To achieve an acceptable error on this metrics we have to go through a platform calibration phase, to have a precise Simgrid platform definition; and a model calibration phase, to correct the eventual scale problem. The goal of the model calibration is to determine an eventual scaling factor to apply during the model generation in order to have confidence that we retrieve real execution time in simulation, whatever the Spark application execution traces that are given to the generator.

**Execution time conservation with different platform parameters:** To be able to do the model calibration we also have to confirm that the bottlenecks of the application are also conserved and that changes on the platform (I/O and network bandwidth/latencies, number of nodes, number of core per nodes, etc.) lead to the same change of bottlenecks in real condition and in simulation.

With the guarantees that this evaluation will bring on the model, we would be able to use it for several other studies:

- Provide decision help for cluster administrators like infrastructure dimensioning, selecting the right configuration tweak...
- Study DFS Daemon interference on HPC jobs.
- Study data movement and staging/prefetch for Big Data application.

### 5.9.3 Multi-tier I/O aware scheduling policy evaluation

Another usage of Batsim is to evaluate scheduling policies on a multi-tier storage platform. The Greco ANR[9] is currently using the I/O aware simulation of Batsim in order to simulate the distributed platform of computing heaters from Qarnot Computing[10]. Note that they have developed a temperature model as a Simgrid plugin that is used through Batsim by the scheduler to take "heat-aware" allocation decision.

### 5.9.4 File System Model Limitations and Evolutions

The file system model described in Section 5.4 has some limitations that are discussed in this section.

First, the storage model is very simple and does not reflect the fine grain behavior of different kinds of storage like HDD, SSD, or NVM. Also, The storage model does not include disk size limitation enforcement, i.e. when a disk is full, and even if this can be done by the decision process, the contention behavior that it implies is not modeled. To overcome the storage model limitation, it would be possible to add a more realistic model into Simgrid, but it may induce large changes in the underneath contention model.

The fact that the file system model is held by the decision process increases flexibility and allows Batsim users to model any kind of behavior, but it has the drawback to lead to multiple implementations of the same model (one for each scheduler). This can be overcome by putting the file system model into an external process, which is even more realistic, but it would increase the maintenance and experimentation cost.

Another limitation is the absence of cache effects in the model, either from the storage itself or from a second level like burst buffers. Also, cluster file systems have metadata servers, which is a common source of bottleneck, but are not taken into account in our model.

One way to allow the modeling of fine grain behaviors like those of the metadata server or the cache effects is to put the file system model inside Batsim. This would

[9] `http://www.agence-nationale-recherche.fr/Projet-ANR-10-SEGI-0004`
[10] `https://www.qarnot.com`

force Batsim users to use our model, but this model may not fit their requirements. Another option is to provide an API for Batsim users so they can implement their own model on top of Batsim file system primitives as a plugin instead of inside Batsim directly.

### 5.9.5 HPC Application Model

The results about mixed HPC and Big Data workload presented in the above section (5.8) are based on realistic Big Data application model but the HPC application model was a simple delay with no effect on the simulated platform. We have tried to use more complex models for this study based on the Simgrid MPI application model SMPI, but having a very precise application model requires a lot of calibration to fit the actual application runtime with the simulated runtime [Hei+17b; Hei+17a]. Moreover, it would dramatically increase the simulation time. Given these constraints, the SMPI model is still very interesting to observe fine grain interferences between applications but for large workloads, it is not relevant.

Some effort has been done to generate more aggregated model from the SMPI traces to get similar interferences effect but without the SMPI complexity. These efforts were not conclusive because the loss of information caused by the trace aggregation makes the contention effect disappear.

Another approach is to monitor real applications on a production cluster in order to obtain realistic applications' resource usage profiles. The first issue with this approach is to get resources usage information from real allocation. We have access to usage traces from the Froggy production cluster at the University of Grenoble obtained with the Colmet monitoring tool [11]. These traces contain resource monitoring per application, but some information is lacking, like sources and destinations for the network traffic. This information is needed to generate a precise communication matrix that is required by the parallel task model. An improved Infiniband monitoring tool, coupled to traditional one could solve this issue. More generally, we advocate for a more precise resource usage tracking in the clusters because the data necessary to generate sound and realistic application model are currently missing.

Note that using production cluster traces has also drawbacks: applications are already subject to network interferences that are not controlled. This directly

[11]https://github.com/oar-team/colmet

biases the extraction model process, which can hinder the model precision. One could extract jobs traces that run in isolated conditions to avoid this problem, but the file system would remain shared and applications would be seen as “black boxes”, which would limit the models that could be generated from those traces.

A solution for both problems is to run representative applications, like Exascale Project Proxy Applications [Pro18], in a controlled environment like Grid’5000 with a full network traffic monitoring. With this approach, we can correlate each application’s internal behavior with their resources usage in order to model them precisely enough to get a sound model that is not too heavy. To do so, the idea is to mimic the Big Data job model by splitting the job into stages in order to keep the resources’ usage peaks (I/O, CPU, communications). This requires splitting the job into meaningful stages, with each stage capturing different resources’ usage patterns, for each kind of application. This is where the knowledge of the application internal will be very insightful.

## 5.10 Conclusion

In this chapter, we have presented our approach to simulate a converged Big Data and HPC infrastructure. This approach is based on Batsim in which we have added an I/O model that allows us to implement a generic file system model along with Big Data application model. We have also reproduced the BeBiDa scheduling which dynamically shares resources between an HPC and a Big Data scheduler on top of the Batsim scheduler protocol using a message broker.

We have managed to produce a working implementation of BeBiDa in simulation. The accuracy of our simulation is backed by the fact that we are able to produce simulation results that corroborate the real condition experiments regarding the Big Data time effectiveness metrics.

Using the file system models we have simulated Big Data application on top of different file systems and in different contexts: one application alone, within a Big Data workload, and within a mixed Big Data and HPC workload using BeBiDa. The experimental results show that for Big Data application alone on a DFS and on a PFS have comparable performance if the PFS network access is correctly dimensioned. But, facing a full Big Data workload, or a mixed Big Data and HPC workload, the applications using a DFS have stable performance whereas the ones using a PFS are highly impacted by the PFS network access contention point.

To conclude, the new I/O aware capabilities of the Batsim simulator allow us to reproduce and study hybrid HPC/Big Data systems and are generic enough to simulate different kinds of file systems. This opens a lot of paths to explore in I/O aware scheduling policies, but also in infrastructure dimensioning, with the capability to find the limiting resource for a given workload.

In future work, we would like to extend our current study of hybrid systems. But, in order to obtain more exhaustive simulation, we need to improve the workload and infrastructure modeling. For the workload modeling, we want to validate our Big Data application model more extensively, and create one or more I/O aware HPC models based on recurrent I/O patterns. Regarding the infrastructure, we would like to improve the file system model by adding the storage hierarchy into the model. Moreover, we would like to expose a new API to Batsim users for them to build their own file system model with better reusability.

# 6 Reproducibility: Scientific Methodology and Tools

## Chapter's Abstract

The production of scientific knowledge relies on experiments. These experiments increasingly rely on software. While science progresses, the daily-used software grows in complexity. This complexity exacerbates the difficulty to reproduce scientific results based on software without a proper methodology.

While preparing this dissertation, we conducted experiments in various contexts. We indeed conducted experiments in both real conditions and by simulation. In this chapter, we first detail lessons learned along with our various attempts to achieve *reproducibility*. Leveraging these experiences, we then shed a novel light on reproducibility by taking into account the software development workflow. We furthermore define an experimental data analysis workflow, and how this workflow integrates with a reproducibility-compatible approach. We then provide a list of good practices to conduct reproducible experiments. We describe how this approach has been implemented for this dissertation, and detail the tools that have been developed and/or used.

We conclude that reproducibility cannot be reduced to a matter of making repeatable science, as it also improves confidence in results by allowing peers to inspect and review the quality of the work. Finally, it is a key point in permitting internal and external researchers to reuse, contribute, and collaborate easily.

## Chapter's Contents

The production of scientific knowledge relies on experiments that increasingly rely on software. In a study by Hannay et al., researchers claim that they spend 40% (30%, resp.) of their time using (developing, resp.) scientific software [Han+09]. The amount of time devoted to scientific software is increasing. Furthermore, an increasing proportion of publications (at least 65%) are backed by software [HB16]. While science progresses, the complexity and number of software stacks in use increases. Such a growing complexity exacerbates the difficulty to reproduce scientific results based on software without a proper methodology.

Among other sciences, Computer Science is young. While other, more established, scientific domains may already have strong methodological standards, our community is still in the process of defining such standards. One may argue that the technology powering computers is so rapidly evolving that it is vain to set up a complex experimental protocol without it becoming obsolete right away. But, as stated by Popper in his seminal book about the scientific method [Pop59]: „non-reproducible single occurrences are of no significance to science“. A study by Collberg et al. demonstrates that most productions in the field of Computer Science are not even repeatable [CPW15]. We stand against scientific contributions relying on disposable and non-repeatable software: **We need to build knowledge in a way that allows others to use it as a basis to construct more knowledge.**

The goal of reproducibility is not only to repeat someone else’s experiment. It is to make scientific work verifiable, and where every step of the experimental process can be inspected in order to be understood and fully or partly reused.

Hence, we need as a research community to tend to a more reliable way to produce experimental results, in order to capitalize trusted knowledge. Because Computer Science is based on software, scientific experiments have to be done with thoughtful experimental methods including the best software engineering practices and tools:

**Methodologies:** Defines our goals, the way to reach them by using the right methods, and standardize this approach.

**Engineering:** Use the most adapted solution to our problem and provide all the necessary material to the community to make experiment reproducible.

**Tools:** Use sustainable and reliable tools to implement these solutions, to permits anyone to repeat and continue the work that has been done.

In this chapter, we propose in Section 6.1 a methodology that consider the experiment software development workflow in order to provide reproducibility and the ability to reuse and modify software. The Section 6.2 defines the experiment results analysis reproduction, and Section 6.3 provides a list of common issues along with good practices to avoid them, with a discussion on how these methods where applied during this thesis. Then, the Section 6.4 gives a list of software tools I have developed, or contributed to, during this thesis and why they help reproducibility. Finally, the Section 6.5 concludes this chapter.

## 6.1 Considering the Development Workflow to Achieve Reproducibility with Variation

The ability to reproduce an experiment is fundamental in computer science. Existing approaches focus on repeatability, but this is only the first step to reproducibility: Continuing a scientific work from a previous experiment requires to be able to modify it. This ability is called reproducibility with Variation.

In this section, we show that capturing the environment of execution is necessary but not sufficient; we also need the environment of development. The variation also implies that those environments are subject to evolution, so the whole software development lifecycle needs to be considered. To take into account these evolutions, software environments need to be clearly defined, reconstructible with variation, and easy to share. We propose to leverage functional package managers to achieve this goal.

This section is an extended version of the article presented at the ResCuE-HPC Workshop, at the SC'18 [MFR18].

### 6.1.1 Motivation

Reproducibility in computer science should be the main concern for the credibility of every scientific contribution.

But, what is the real purpose of reproducibility? It is the capitalization on the work of all the people of the scientific community, to move forward scientific knowledge; so the science is not restarting from scratch every time that a scientist disappear. It is common that a project dies because the only people that own essential unwritten knowledge are gone. This problem can be evaluated with the "Truck Factor" metric. This metric is defined as the minimum number of contributors that have to suddenly disappear („hit by a truck") before the software project cannot survive. A 2016 study shows that over 133 popular projects on GitHub, 65% of them have a Truck Factor of at most 2 [Ave+16].

Because Computer Science is based on software, it is possible to avoid this issue with good software engineering practices, such as composability and good documentation. But, while the science evolves, simulations become more accurate, and the scientific software stack becomes more complex. Thus, even if the project is documented, going from source code to runnable experiment, i.e. the building process, can be tedious. Moreover, good software engineering is not recognized as it should in Computer Science research community: there is no incentive to take the time to do it right. So, reproducibility in Computer Science needs to be backed with new methodologies, good practices, and appropriate tools. This will allow to build and transmit scientific knowledge more efficiently.

Reproducibility is a wide notion that needs to be specified. Feitelson [Fei15b] has defined a taxonomy of the different way to reproduce scientific results. In this taxonomy, The first level of reproduction is the **Repetition**, i.e. do exactly the same experimental process to obtain the same results. The second level, **Replication** is similar but the experience's input is changed.

Currently, most of the reproducibility tools only support these two levels by capturing the software environment. Indeed, a software environment is hard to reproduce, and without it, it is impossible to run the experiment. Also, the software environment of an experiment tightly depends on the Operating System (OS) distribution it was built on. It is sometimes impossible to install it on another distribution because of interdependency issues. One approach to solve this problem is to snapshot the software environment into an image. But, even if an image of the experiment runtime environment is provided by the original author, continuing his work requires more than just repeating the experiment; to be able to corroborate someone's approach, we not only need to rerun experiments, but we also need to modify them: test new variations, add more parameters and develop new features. This is the next level of reproducibility in the aforementioned taxonomy, called **Variation**.

Enabling scientists to reproduce an experiment with variation requires that the reproducer is able to rebuild experiment software with some modifications (even if the software is unmaintained and the tools necessary for building it, are long gone). It means that the reproducibility with variation can be achieved if the reproducer is able to reproduce not only the experiment "production" environment but also the “development” environment which is necessary to modify the software. Moreover, when a variation of a previous experiment produces new results, this experiment should also be reproducible.

In this context, we are proposing a new way of seeing reproducibility through the scientific software development lifecycle. Each step in this lifecycle requires a software environment. We define a software environment by a set of applications and libraries, with all their dependencies, and their configurations, required to achieve a step in a scientific workflow.

### 6.1.2 Software Development Workflow and Reproducibility

In computer science, a scientific workflow contains a software development lifecycle that starts by setting up a development environment with build tools and dependencies. Then, this environment is used to build a production environment that will, in turn, be used to run the actual experiment. But, software development is an iterative process: one can produce multiple versions of the production environment, or even modify the development environment by updating or adding tools. This process is in the middle of the scientific workflow. All the produced software environments, either development or production environments should be captured to enable reproducibility. The Figure 6.1, exhibits that the first two levels of reproducibility can be achieved with only one environment, but **reproducibility with variation requires taking into account the whole development process.**

Scientific workflows themselfs have to be reproducible: Thus, internal (e.g., colleague, intern) and external (e.g., scientist from another laboratory or reviewer) contributors have the capability to reproduce them and moreover to contribute them.

But it is hard to keep track of all the tools that are used in a development journey: It can be documented at some point, but with the subsequent development iterations, the actual environment drifts away from its documented state. Moreover, most of the developer’s systems have pre-installed tools that turn into hidden dependencies when the documentation of the development process omits them.

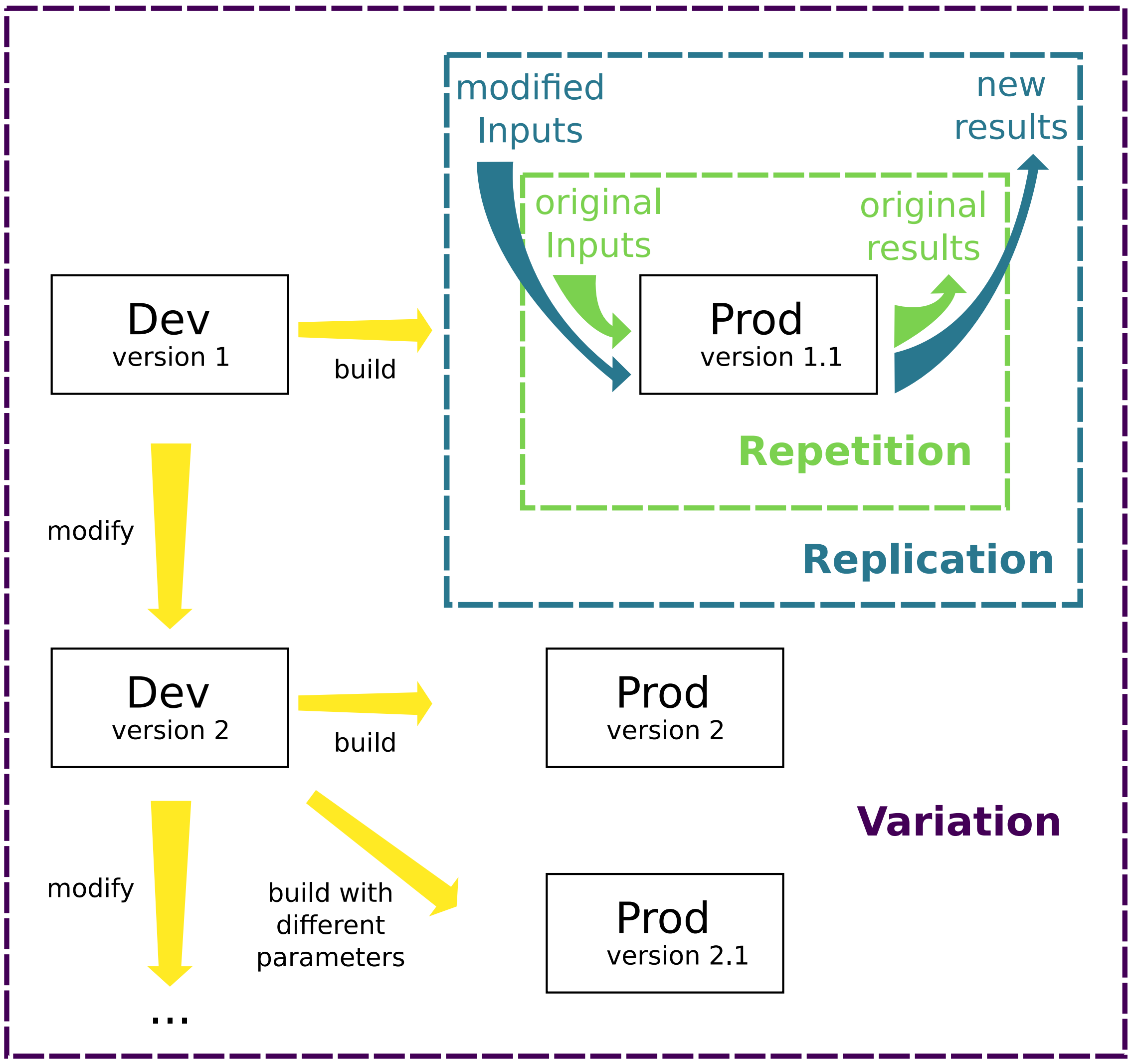


Figure 6.1.: The various levels of reproducibility in regard to the development lifecycle: Variation requires to enclose the development environment and provide a way to modify it while keeping reproducibility.

So, to provide this capability the development environment should be defined entirely, and every change should be tracked. This statement also holds for other software environments involved in the authors' workflow, as the ones used for input data generation, or output data analysis.

### 6.1.3 Reproducible Software Environments with Nix

Courtès et al. emphasize that functional package managers (FPM), like Guix and Nix, are good candidates to share complex and upgradable environments [CW15a]. In the following of this section, we will focus on Nix, but most of the assertions also hold for Guix. FPMs apply the concept of functional programming to software packaging. The building process of each software is described through a function. The dependencies are also functions that are given as inputs. This function, or package definition, allows to precisely describe a package: where and how to gather the source code, which exact version to use, the dependencies and their versions, and finally how to build the package. When a package is built, the dependency graph is resolved by a lazy evaluation of the function parameters, and all the necessary piece of software are built as needed. The result of the evaluation of a package definition is called a derivation. A derivation is concretely a set of files that contains the results of the building process of the software, which placed on the special place that contains all the derivations: the store. Finally installing software is simply exposing a derivation from the store through symbolic links. Nix packages are written in a functional Domain Specific Language (DSL). This ensures that each build is pure, i.e. it only depends on its inputs, and the same inputs give the exact same package even on a different machine.

To implement the workflow described in Section 6.1.2 with Nix, the critical feature is the capacity to create a software environment without virtualization. This feature is used to create an isolated environment for the package building process. An environment can be seen as a set of derivations, and relies on the fact that an FPM can infer all the dependencies of a derivation, and only expose these dependencies on the specified environment. Installing the environment will expose the packages described in the environment to the user. Archiving an environment will extract the whole dependency tree of the environment, and create a self-contained archive. The resulting tarball, also called a closure, contains every binary and file necessary to run the packaged applications. Thus, the environment can be installed on another machine without any external download or building process.

To achieve each level of reproducibility with Nix, the first requirement is to create a package definition for each application and its dependencies. Thankfully, Nix is a very active community and more than 40000 applications are already packaged in the main repository called “nixpkgs”. It is also possible to maintain a private set of packages that import dependencies from “nixpkgs”.

The first level of reproducibility, the Repetition, can be achieved by providing to an external scientist the production closure, that can be installed to repeat the experiment. The environment could also contain all scripts to deploy and run the experiment.

The second level of reproducibility, the Replication, that consists of replaying an experiment while changing its input, have the same requirement as the repetition: Only the production environment is needed.

The third level of reproducibility, the Variation, is where using Nix is the most advantageous. Nix provides the interesting feature, called the “nix-shell”, permitting to enter the package build environment. Hence, packaging software with Nix have the side effect to provide also the build environment for the users. Nix capacity to define software environment and software package in a unified way gives the scientist the ability to share a reproducible production environment, and the associated development environment, with a single definition.

### 6.1.4 Related works

The Popper method, proposed by Jimenez et al., describes a structural framework for dependencies and artifacts [Jim+17]. They identified a generic workflow describing an experimental methodology, from source code to the final manuscript of a contribution. Our contribution is compatible with this approach. The Popper method proposes a structural organization of the experiment, whereas our approach proposes to implements a part of this workflow using Nix.

Repeatability has been the focus of previous works on reproducibility. The platform presented by Ricci et al. [Ric+15], has the ability to instantiate an experiment environment in their infrastructure from a previously captured environment. This approach is interesting as it provides a way to repeat experiments that require specific hardware. Our approach could be complementary to cover both hardware and software, and provide a higher level of reproducibility.

Boettiger et al. propose to use the container technology Docker in order to improve reproducibility. They also introduce the notion of development environment [Boe15]. From our implementation with Nix, the docker approach shows similarities. However, Nix closure is more adapted than the Docker images for application packaging because Docker provides an inappropriate level of abstraction: Docker is about constructing and configuring a complete OS, instead of declaring application dependencies.

The Chapter 3 contains comprehensive description of the methods that can be used to package an environment on HPC systems with reproducibility in mind.

Another way is to capture a software environment from an experiment execution. This is the approach of ReproZip [Chi+16]. A complementary approach is to generate development environment from a project workspace, i.e. detect the project language and build an environment containing the dependencies described in the per-language standard package manager or infer from the code itself and external metadata [Ste19; Lab19; Pro19b; Ali19; Ope19]. Both approaches, environment capture and environment generation, have the advantage to eliminate the hassle to think about reproducibility from the beginning of a project because it can be applied afterward. But, these approaches are limited in scope: the capture approach only handles simple cases, where an experiment is based on a single process, and the generation approach is only working for some language and for already packages dependencies. Furthermore, they do not provide environment reconstructability, and thus hinder reproducibility with variation.

Constructing a reproducible experiment and workflow with an FPM has already been explored. In [Wur+18], they build a toolset upon an FPM (GUIX), to facilitate the usage of bioinformatics common pipelines. They argue that using an FPM is a good foundation for reproducible computational experiment workflow.

The Blue Brain project [DDS15b], is a big project, with a complex software stack, that aims to model a mammalian brain with a computer. In addition to a structured development workflow (using git, agile methodologies, code reviewing), they decided to package their workflow with Nix. They identified nine properties that are facilitated with Nix, from reproducibility to deployment and cross compilation (compiling software from one architecture to another). Their approach is based on internal needs and specific use cases, whereas our contribution focuses on the role of the development lifecycle in the reproducibility, with Nix as a possible implementation.

### 6.1.5 Discussion

The proposed workflow is not considering input and output data. One approach is to package the experiment data inside the production environment. It is a viable solution for a small amount of data, but most of the experiments have managed data separately with other tools. In simple cases, where the amount of data is small, it can be handled with Git directly into the experiment source code repository. Git extensions like `git-annex` [git18b] or `Git Large File Storage` [git18a] are more appropriate for larger files.

The data related to the experiment is not only input and output data, the software environments that are used to develop and run the experiment also have external inputs, like the source code of the experiment, the dependencies, the build tools binaries, and the configuration files, i.e. all the other artifacts necessary to build the environments. With Nix, it is easy to extract these artifacts for the experiment production environment. Nix is capable to export all the dependencies of an environment in a closure that can be imported on any machine where Nix is installed. This feature of Nix is very hard to achieve with other kinds of tools which suffer from the lack of clear dependency definition. However, even if it is straightforward for the production environment, how to extract this closure for the development environment is unclear for now.

Capturing and archiving the environments closures is necessary for the Variation reproducibility because the lifespan of Internet links is only a few months [Law+01]. Even if Nix is capable to rebuild everything from source, the source code repository can be unavailable, breaking the environment reproducibility. The problem that emerges is that closures also have to be safely archived, versioned, and accessible for a long time period. A mid-term solution would be to store those closures (or even replace the closures content) using perennial archives such as Internet Archive[1] and Software Heritage[2].

Nix itself has limitations, its usage requires to understand the concepts behind the FPM and to learn the Nix DSL. Also, even if the Nix build system provides the framework to achieve reproducible builds of packages, the bit-wise reproducibility depends on the way the software itself is built. For example, one may have to patch the Makefile.

[1]`https://archive.org`
[2]`https://www.softwareheritage.org/`

In the case where the experiment results depend on the OS Kernel, (e.g., performance evaluation) this approach is not sufficient. Indeed, Nix packaging handles the whole application software but not the OS Kernel. So, the proposed workflow needs to be supplemented with a building process definition of the entire OS — and not just the application layer — to be able to reconstruct a complete OS image with variation. A Linux distribution based on Nix, called NixOS, is a good candidate for such a purpose. Using Nix DSL, one can define a complete OS programmatically. It thus provides the same advantages as the packaging: reconstructibility and portability.

Another compatible approach with the previous one is Kameleon [Rui+15]. This approach let user extend template recipes or write from scratch its own environment recipe. The user can build an environment with an interactive breakpoint triggered in case of failure. Kameleon also features a cache which try to store all the external dependencies and checkpoint/restart to avoid to rebuild every steps of a recipe in case failure. It is made to create OS images from a recipe that describe every step of the process. It is more generic than NixOS because it can be used to create an image of any kind of Linux distribution on any architecture, and the software appliance produced can be exported as any format (Virtual Machine, Docker container, a tarball, an Ext4 CPIO, etc.) in order to be deployed anywhere. More details on Kameleon are given in the Section 6.4.1.

When specific hardware is necessary to achieve reproducibility, an additional layer of control is needed. Testbeds like Grid'5000, Chameleon, and Emulab, are giving this level of control with the capability to create and deploy OS images on the fly on different hardware.

### 6.1.6 Conclusion

Reproducibility with variation is the next level of reproducibility that the Computer Science community should aim at. The Variation requires taking into account the software development workflow, including the capability to modify and rebuild environments. The use of functional package managers is a promising approach. This kind of tool permits to achieve this last mile to the reproducibility with variation, with a unified way to describe environments and packages, and a simple method to backup and to restore them.

## 6.2 Towards Reproducible Data Analysis

In the previous section, we have discussed the reproducibility with variation only for the experiment itself. But, the scientific research process also includes lots of work before and after the experiment: the data analysis.

In fact, most of the published paper contains only a subset of the whole analysis that was made during the scientific process of writing a paper. It is really interesting, in order to understand a contribution, to repeat, and reproduce with (or without) variation the data analytics process that drives the scientific thought. A recent study on a set of published paper in the geographic information science field nominated for best paper shows that none of them are fully reproducible [Nüs+18]. These papers are data-oriented, and thus, based on a data analysis that turns out to be non-reproducible for multiple reasons: Missing input data, undefined preprocessing, undefined method/analysis/processing, missing computational environment, and missing results.

Based on these observations, we have defined different levels of reproducibility for the data analysis with their requirements, in order to give concrete evaluation criteria for reproducibility and to advocate for better reproducibility of this part of the scientific work.

### 6.2.1 Definition of the data workflow

The following definitions are defined the different inputs and results type that can be found in an experiment workflow (see 6.2):

- **Raw inputs**: Inputs needed to conduct the experiment. Sometimes too big to be kept.
- **Processed inputs**: The formatted and/or filtered data from the raw inputs to be given to the experiment software.
- **Raw results**: Results produced by the experiment.
- **Processed results**: Formatted and/or filtered data from raw results to be given to the analysis tools.

- **Analysis results**: The graph, table or any form of high-level results that could be included in a scientific publication.

In the previous list, each element depends on the previous one: It represents a data workflow example of an experiment. This workflow may be simpler or much more complex. But, going from one element to the other is a specific process that needs to be documented and reproducible. Depending on the level of reproducibility you want to achieve, you may avoid storing each intermediary data because you are providing the process to reproduce them from the previous step in the data flow. Note that, if the process to obtain intermediate data is long and/or complicated, it is always interesting to archive them in order to avoid the hassle of reproducing them even if you provide the process to do so.

In the following of this Section, we propose various reproducibility levels, from simple result analysis to full reproduction. These levels have an inclusion relation between them. It means that if someone can reproduce the experiment it can also repeat it and reproduce the analysis.

### 6.2.2 Repeat analysis ($R0$)

Repeat the same analysis from experiment results. It is an intellectual process that requires no specific tool, only the data: tables and graphics provided by the original authors of a paper, in addition to the scientific though narration that explains the analytic process.

This is the same as reading the paper itself but with more details on the analysis, that permits to fully understand the choices of the authors and the fine grain analysis that may be too long to fit in the paper. This can be provided in a technical report associated with paper, or in a notebook export.

Requires:

- Analysis narration
- Analysis results

### 6.2.3 Reproduce analysis with the same inputs (with variations) ($R1$)

It represents the reproduction of the analysis results using the same inputs. Reproduce an analysis from some experimental results with or without variation has the same requirements, so these two levels are mixed.

This type of reproducibility is highly valuable for reviewers that want to understand fully an analytic process, and thus can make suggestions on what to include and or exclude from the results presented in the final paper.

To achieve this level of reproducibility, one needs to interact with the data. Hence, a static format like the technical report is not sufficient. A popular solution is to use Literate Programming [Knu84] for that purpose, in particular using interactive notebooks.

Requires:

- Original input data
- Analysis narration
- Analysis results
- Analysis software environment (modifiable)

### 6.2.4 Reproduce analysis on other inputs (with variation) ($R2$)

This level of analysis reproducibility requires to be able to change the input data, therefore, to obtain new input data. This can be done by getting more inputs from the original authors, or by rerunning the original experiments, or even by generating new data from another set of experiments based on another implementation.

It is necessary for a scientist to be able to corroborate (or not) the work of others without re-doing the whole analysis process. Using the analysis process of the previous research permits comparing more fairly by avoiding some bias, like the tendency to show only the part of the results fitting his point of view. The new analysis can also be extended and the original authors can then reuse this new piece of analysis to enhance their future work.

Requires:

- Original input data (for comparison)
- New input data
- Analysis narration
- Analysis results
- Analysis software environment (modifiable)

### 6.2.5 Conclusion

The presented 3 levels of analysis reproduction obey a property of inclusion:

$$R0 \subset R1 \subset R2$$

Indeed, while $R0$ (which is the common level proposed today, if any) only requires to publish a separated analysis, $R1$ need to use interactive tools, and $R2$ add to these requirements the possibility to ingest new data to be able to compare.

We advocate that the Reproducibility with Variation discuss above, represented here by the level $R2$, should be aimed by the computer science research community. It would improve results reliability, increase scientific corroboration, and creates new collaboration opportunities.

## 6.3 Good Practices for Reproducible Experiment

Even if computers are seen as deterministic, computer hardware is dependent on their physical environment like heat variation, vibrations, magnetic fields and so on. Hence, computer performances and behaviors suffer from great variability. Fortunately, thanks to a lot of efforts from hardware vendors and operating systems, the determinism of computation can be achieved, but the performance is still unpredictable without full control of the machines' environment. This is one of the main barriers to achieve experiment reproducibility in computer science, especially when it involves performance evaluation.

In the case of simulation, where performance evaluation is not an issue, the result can be deterministic with respects to the simulations inputs parameters and input data, thus independent of the hardware it was run on.

For experiments in real conditions, also called in situ, the constraints are larger because the operating system and the hardware needs to be controlled. However, in most computing center this control is not given to the end-user.

Whatever kind of experiments, reproducing an experiment requires that the input data is provided by the original authors, along with the software environment, and the hardware description if needed. This study shows that only running the code of an experiment is already a challenge [CPW15]. The causes of this lack of knowledge sharing can be numerous:

- There is no public data available apart from the paper.
- Restrictive license or any intellectual property problem.
- The unreliability of "home made" software.
- Missing experimental plan: used parameters are not provided.
- Input data and/or results are not provided.

- Lack of access to the experimental environment (like testbed or computer Grid).

Despite the intellectual property problem, which by definition prevent the reproducibility, all the aforementioned problems could be addressed by some good practices and appropriate tools:

1. P1: Use a long-term, publicly available, properly organized, version control repository.
2. P2: Use reusable open source software and proven simulators.
3. P3: Provide software environments.
4. P4: Provide experiment plan, design, and workflow.
5. P5: Provide inputs and results.
6. P6: Give a full description of the hardware used and free access (if possible).

These properties are described more thoroughly in the following sections.

### 6.3.1 P1: Use a long-term, publicly available, properly organized, version control repository

One of the main problems with repeating a software-backed experiment is the access to the software itself. Basic software project management applied by most of the current projects is to use a version control system (VCS). This VCS should be reliable with long-term support and public access. If not, a third-party archive system like Software Heritage can be used. Sowtware Heritage proposes to store a snapshot of your project, with its entire history, and to freely host it with all the good properties listed above.

But using a VCS is not enough to give to a stranger an idea on how to use experimentation code. We need some kind of standardization on the organization of the code inside the repository. For classical software projects, most of them are following a classical pattern dictated by the language ecosystem and the associated tooling. But for software experiment, we need to distinguish experiment, to store material for each experiment, to find analysis scripts that produce the graphs, and so on.

In [SLD15] they propose an approach based on a VCS workflow, here Git. The idea is to use Git branching mechanism to organize experiments described in an Org-Mode notebook. This approach has the advantage to organize the results' data versioned in Git while tracking the moving implementation of the experiment, but it relies on a work discipline that is hard to maintain.

The Popper Convention [Jim+17] is proposing an organization convention for experiment repository, so it is easy to link every step of the experiment process, i.e. from the input data to the paper. This convention also encourages applying DevOps methods [Wik19b] to scientific experiment by automating every step. Thus, the experiment can run through a continuous integration system (CI) and clearly present which experiment was successful and unsuccessful with the code of this experiment directly attached to it along with experimental results. While this approach can be complex to set up, especially when using resources reservation (e.g., Grid'5000 testbed), it brings a lot of confidence on the experimental results thanks to better traceability.

### 6.3.2 P2: Use reusable open source software and proven simulators

First of all, closed source software hinders reproducibility by nature. Even if they are gracefully provided by their owner, they are not observable and can only be seen as black boxes. This fact totally invalidates the possibility to reproduce experiment with variation. But, some level of reproducibility can be reached by following the other best practices proposed in this section. For example, the analysis part can still be reproduced if the data and the analysis environment are provided by the authors.

Lots of published paper made a simple simulator that is used only for the paper or chooses an on-the-shelve simulator that is not proven to be accurate. In order to validate simulation results the simulator must be assessed theoretically and if possible experimentally.

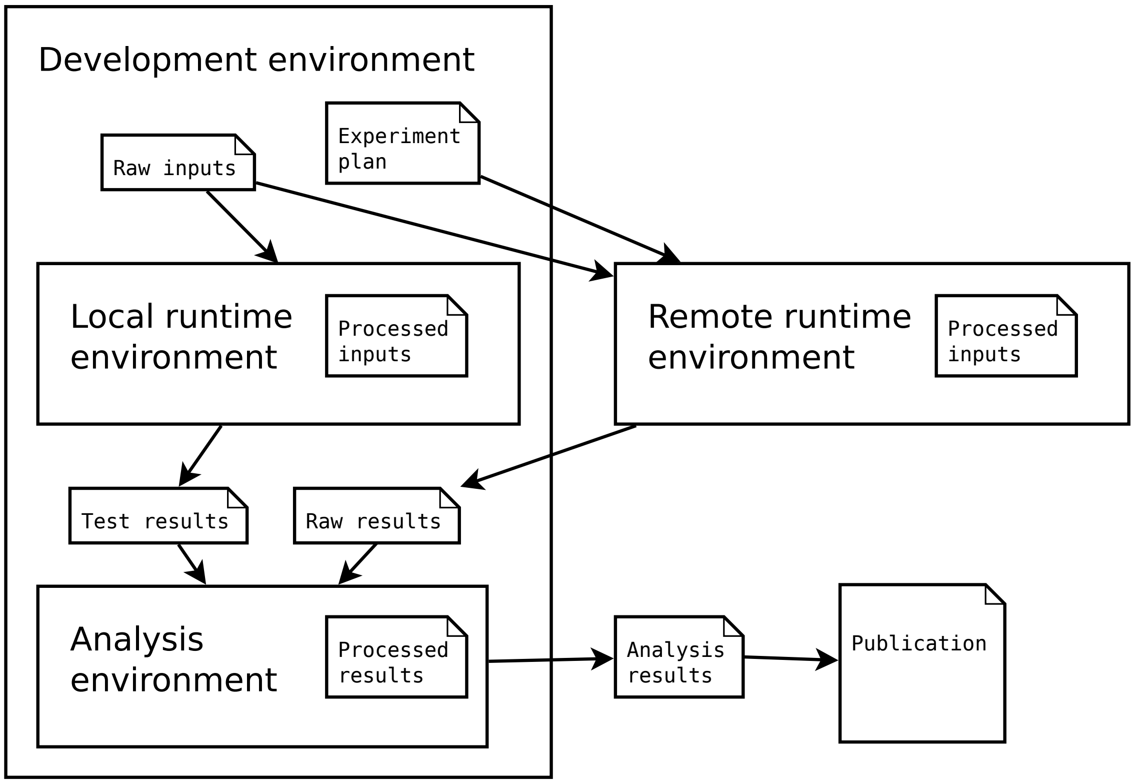


Figure 6.2.: Environments and inputs in the experiment process. All these elements must be provided to achieve full reproducibility.

### 6.3.3 P3: Provide environments

First, what is an environment in this context? An environment can be defined as a set of tools and their dependencies packaged together. It is self-contained and provides the ability to run without any users'action.

As shown in Figure 6.2, various environments are required to achieve full reproducibility:

- **Development environments**:

  The environment where the experiment is developed. It contains the tools used to build the experiment tools. It is possible to have one development environment for each environment depending on the experiment setup.

- **Runtime environments**:

The environments where the experiment are launched, containing experiment dependencies and input materials. It is possible to have multiple runtime environments. For example, a local environment to test the experimental setup and another environment on a remote cluster to actually produce the results.

Note that input materials may be dynamically loaded from a remote repository, but it has to be accessible from this environment.

- **Analysis environment**:

  The environment where the results' analysis is done. This environment must contain or have access to raw results, results' metadata, and processing script. The latter can rather be included in a runtime environment if it is more convenient.

Figure 6.2 provide an example: the development environment is the scientist local computer (e.g., a laptop). The depicted setup uses two runtime environments: a local environment for testing and a remote environment running on a dedicated device like a laboratory server or a testbed. The analysis environment is also located on the local machine. This is just an example: this configuration may vary according to the computing resources necessary for the experiment, the inputs, and the results size and complexity.

The general idea is that any software that is used at anytime in the experiment process should be provided with its environment.

The major problem is: how to share these environments? On a local machine you can create an installation script to install and build the tools but what about your operating system (OS) and the tools dependencies? And what if someone likes to test variations on some software components?

The only way to be sure that the exact same experiment can be run again and reproduce with variations is to provide a full environment: a complete software appliance including the OS if necessary, and the process to build this environment in order to meet data provenance requirement.

Section 6.1 discusses this matter in more details.

### 6.3.4 P4: Provide experiment design and workflow

If a reviewer would like to reproduce an experiment that has complex execution and analysis pipeline, he or she needs to reproduce the scientific workflow of the original author. The problem is that, if the workflow is not documented, the reproducer needs to guess dependencies between process parts, which is error-prone and may lead to unreproducible results.

So, the experiment workflow needs to be documented, but the final experiment process is just a small part of the whole scientific reflection. To keep track of this reflection process, it is important to have a narration coupled with the experiment workflow documentation. Notebooks can contain this narration and even the workflow itself using the concept of literate programming: melting formatted text with code and results to provide a runnable laboratory notebook.

Lots of notebook technologies have emerged recently, and one of the most advanced is Orgmode [org19]. It is an Emacs [Pro19a] mode that features highly customizable text formatting in the OrgMode syntax and code editing in any language. When a notebook is executed it can produce a full paper with the text and the figures, while keeping track of the code that was used to generate the figures. Jupyter Notebook [Jup19] draws a lot of attention lately in the scientific community, and as a web-based technology it is much more accessible to newcomers, yet less powerful than Emacs Orgmode. Lots of other similar tools exist that are more or less suited depending on the workflow and the preferred language (Rstudio/Knitr, Sage, Mathematica, nteract, Beaker, Apache Zeppelin, etc.). Whatever the tool used, documenting a scientific process through and the usage of notebooks is a big step towards reproducibility of the analysis process described in 6.2.

With runnable laboratory notebooks, the experiment design and the analysis workflow can be safeguarded, but the rest of the experiment workflow also needs to be documented and if possible in a runnable way. To be executable, and thus repeatable, this workflow needs to be encoded in a language. A lot of workflow languages exists, e.g. one of the most advanced projects is the Common Workflow Language [CWL18], that is already implemented in a lot of tools and used in the Bioinformatics community.

The experiment workflow is highly dependent on where the experiment will run. A good practice to improve reproducibility it may be to abstract the experimental platform in order to be adapted to any other platform easily. But, the major problem is to have access for a reproducer (and even the original author) to the resources needed to rerun the experiments. If these resources are only software, the problem can be solved using the mechanism explained in Section 6.1. If the needed resources include a specific hardware setup or specific network topology, it may not be available anywhere outside of the original experiment platform. The experimental workflow may also depend entirely on the capacity of the experimental platform, like the capacity of Grid'5000 to deploy a specific OS Kernel. In this case, using a dedicated workflow script is more suitable because it can encode the whole scientific workflow, including fine-grain resource selection and reservation [Gri18].

### 6.3.5 P5: Provide inputs and results

The original inputs data are crucial in the process of reproducibility. The study of this field is call *Provenance* [HDB17]. Provenance tracking can apply to any objects or artifacts that is an input of the experiment. It can be raw and processed data, experiment raw and processed results, but also any software involved in the experimental process. Note that the capture of the software environment is already discussed in details in the Sections 6.1.

The Section 6.2 expose the different levels of reproducibility we can achieve regarding the data and the analysis workflow. But, once you have determined the level of reproducibility and thus the piece of data that needs to be conserved, you need to find a suitable way to archive them.

In order to enforce data provenance, every archived object should be stored in a long-term, public, and reliable place, with a permanent way to refer to any piece of data.

The simplest way of doing this is to use the VCS repository that already contains all the experiment design (See P1 in Section 6.3.1). Once archived in Software Heritage, you can have good confidence in the durability of the data. But, it is not convenient to reference a particular piece of data in the middle of a large Git repository.

So, important dataset can be stored using other means: dedicated Web platforms like Figshare [Fig19] or Zenodo [Zen19], or Peer-to-peer based sharing like Academic Torrents [LC16], that are providing a lot of useful features (DOI, Metadata, search tools, Bibtex, . . . ), increasing visibility and permitting to other researchers to download and cite datasets with ease. Academic Torrent has the advantage to be unlimited regarding the size of the dataset (the biggest one is 4.79 TB), whereas Figshare and Zenodo are limited to 5GB and 50GB respectively.

The same kind of services can also be found in online journals like PeerJ [Inc19], with automatic creation of DOI for each part of the uploaded articles. It also has the possibility to store supplemental dataset but it is limited to 50MB only.

### 6.3.6 Reproducibility of this thesis

During this thesis, we have tried multiple strategies and tools to achieve reproducibility, and thanks to the accumulated experience we have elaborate the guidelines proposed in the previous section (6.3). In the meantime, we have followed as much as possible our own principles. This section is a description of the methods and tools we have chosen to achieve this goal.

#### P1: Organized Git repositories

All the work that was done during this thesis was captured in a version controlled system (VCS) hosted on the Inria's GitLab. These repositories are publicly available, and will remain so even if Inria stops to maintain its Gitlab instance. This is due to the fact that every public repository on the Inria Gitlab is automatically archived by Software Heritage.

Two repositories[3]contain the description and implementation of the experimental work done for Chapters 4 and 5.

Both repositories are organized in a strict folder hierarchy inspired by the Popper convention [Jim+17]. They are organized as follows:

[3] The repository located at `https://gitlab.inria.fr/mmercier/bebida` contains the real-condition experiments.
The repository located at `https://gitlab.inria.fr/batsim/bebida-on-batsim` contains the simulation-based experiments.

**environments** Contains the software environment recipes that are common to multiple experiments. This folder may not be present if each experiment needs its own environment. In this case, each experiment sub-folder contains its own environment description.

**experiments** Contains one folder per experiment, each of them containing one or more notebooks, script, and all the material necessary to run the experiment and its analysis. Materials that are common to multiple experiments are directly located here.

**paper/presentation** : Contains all the necessary material to compile the paper (presentation, resp.) including bibliography and graphics.

### P2: Trusted and open software

All the tools that we have developed and used are open source, thus modifiable. Other scientists are able to build on top of these tools if they want to.

In order to have trustworthy results about computer distributed infrastructure simulation, we chose to base our work on Simgrid [Cas+14]. Simgrid is a large scale distributed systems simulatior. It is widely used, in more than 200 scientific materials [Leg18], and provides computation and network models that are experimentally tested. To simulate infrastructure and resources and jobs management systems (RJMS) we build Batsim [Dut+16] on top of Simgrid. It aims to improve repeatability in this domain that is widely hit by the problems mentioned before, by allowing any event based scheduler implementation to run on top of it. More details on Batsim is Section 6.4.2.

### P3: Software environment for every step

The large experiment done for the Batsim validation (Section 6.4.2) and for the BeBiDa real conditions experiments (Section 4.5) were carried out on the Grid'5000 testbed [Bal+13].

Grid'5000 proposes unique features. On can run software as a privileged user on a shared cluster, and even deploy a custom operating system on the reserved nodes.

It allows to fully control the software stack from the kernel to the applications. For our experiment we have constructed a system images using Kameleon [Rui+15] which is described more thoroughly in the Section 6.4.1. You can review the experiment environment building recipes and reconstruct them in the BeBiDa code repository which is publicly available[4].

The simulation experiments and the results' analysis were performed in a software environment defined with Nix in both repositories. As explained above, there are Nix expression files in all the experiments' folder hierarchy and even in the paper and presentation folder. All these files are describing installable or disposable software environments base on the Kapack repository of package (See Section 6.4.4). These environments are made to be fully reproducible from their description file by freezing all the version of all the software present in the stack and by never relying on packages already installed on the local system.

#### P4: Workflows are defined in Python scripts

In this thesis, we used Execo [Imb+13] to script real condition experiments on Grid'5000. The reusable parts of these scripts are publicly available for the Grid'5000 community (see Section 6.4.5).

The rest of the workflow is described in text files that explain in which order and how to launch each steps. Jupyter notebooks contain the implementation of these steps along with their software environments. Examples of notebooks, one that runs a set of simulation [5], and another that analysis the results [6], are available on the experiment repository.

The software environment required to execute these notebooks is also defined as a nix shell in the same repository[7].

With this separation of concerns between the simulation and the analysis, it is easy to reproduce either the whole experiment or only the analysis part.

[4]`https://gitlab.inria.fr/mmercier/bebida/tree/master/environments`

[5]`https://gitlab.inria.fr/batsim/bebida-on-batsim/blob/master/experiments/bebida-exp-replay/Bebida_on_batsim_simulation.ipynb`

[6]`https://gitlab.inria.fr/batsim/bebida-on-batsim/blob/master/experiments/bebida-exp-replay/Bebida_on_batsim_analysis.ipynb`

[7]`https://gitlab.inria.fr/batsim/bebida-on-batsim/raw/master/experiments/shell.nix`

### P5: Data versioned and published

The input and output data is versioned in the aforementioned Git repositories. Also, some of the intermediate results are in version control, and if they are not, the process to obtain them is documented in the notebooks.

One of the dataset that was generated in during the Batsim workload generation process, which contains a set of MPI applications detailed traces was published on the Zenodo platform[8] and on Academic Torrent[9].

Another set of traces but used for BeBiDa experiment is available on the Zenodo platform[10].

### P6: Hardware description and Open access thanks to Grid'5000

As mention above, the real-condition experiments were conducted on Grid'5000 which provides complete hardware and firmware description of each machine through an API. For example, the API provides the complete description of the Graphene cluster that was used the BeBiDa[11].

Also, Grid'5000 has an Open Access program that let any researcher freely access the platform for two months[12].

This policy permits anyone to access the exact same hardware that was used for the experiments.

### General feedback and thoughts

During this thesis, we have gathered a set of tools that really helps reproducibility but also day-to-day work.

[8] https://zenodo.org/record/45810
[9] https://academictorrents.com/details/53b46a4ff43a8ae91f674b26c65c5cc6187f4f8e
[10] https://zenodo.org/record/2555074
[11] https://public-api.grid5000.fr/stable/sites/nancy/clusters/graphene/nodes.json?pretty=1
[12] https://www.grid5000.fr/mediawiki/index.php/Special:G5KRequestOpenAccess

With literate programming for experiment narration in OrgMode [org19] or in Jupyter notebooks [Jup19], we are able to track the scientific pathway and enable the peer reviewing of it, which is very valuable for students and advisors.

We also have seen the advantage of having Kameleon images that can be deployed on Grid'5000 months after their creation without any problem. It was also simple to slightly modify the image, for example, to update certain tools inside, by only increasing one value in a variable and rebuild. But still, this process takes time, so we have learned that the use of Kameleon should be reserved to the static and system level part of the experiment software environment, i.e. if in your experiment the version of your software is a parameter, do not put in the image; if this software is the kernel then you don't have a choice. Hence, for the dynamic part of the environment (if it is not the part of the system) the use of Nix give a lot of flexibility and guarantees on reproducibility.

We also find out that every step of the input data generation, preprocessing, and analysis should be self-documented in one or more notebooks with the associated environment in order to be reused by others.

More generally we have experienced reproducibility in multiple contexts and it is important to understand that each problem class requires a different set of tools. For example, experimenting in real condition requires to properly automate every step in a script because it has to be launched through a Batch scheduler. Notebooks that are made for interactive work are ill-suited in this case. In the other hand using a script for data analysis lack of flexibility and because unlike notebooks one is not able to write rich comments in-line and can't execute part of the code out-of-order.

## 6.4 Tools

Good science requires good tools. From mathematical tool to simple utilities, all kind of software assists scientists to achieve their goal: creating scientific knowledge. As discussed in the previous section, reproducibility is the cornerstone of scientific construction and creating reusable scientific tools is an important part of this process.

During this thesis, I have developed and contribute to different tools. This section is listing these contributions, their usage in the context of this thesis, and their scientific impacts.

### 6.4.1 Kameleon

Kameleon [Rui+15] is a tool that generates reproducible customized Operating System (OS) images in any format. The focus of Kameleon is to provide an entire software stack, from the bootloader to the application layer, in a fully reproducible manner. It is made to create reconstructible images from a recipe that document the whole building process. This recipe written in YAML, can be shared easily, and be modified at will. Kameleon is divided in two part: the engine[13] that executes the recipes, and the recipe templates[14] that provide a basis for user-defined recipes.

More formally, Kameleon manipulates the concepts of software environments and of execution contexts. A software environment is a set of applications and libraries bundled together in any format. This environment creation process can be defined by a base environment (that can be empty), and a succession of steps that modifies it incrementally until the final environment is created. A Kameleon recipe contains all these steps that are executed in a deterministic order. One important notion in Kameleon is the execution contexts. Indeed, the software environment building process involves multiple execution contexts: Where Kameleon is executed called the “local” context, where the new environment is created and bootstrapped called the “out” context, and inside the newly created environment once it is bootstrapped, the “in” context. Kameleon provides a generic way to define how to connect to each context and how to share information between them. Using this mechanism, a Kameleon recipe is capable to describe any OS building process. In the recipes, this process is divided into three phases:

1. the **bootstrap** phase that involves the “local” and the “out” context to create the basis of the new image, and to enter the new OS. This phase can use any kind of virtualization technique (chroot, Qemu/KVM, Virtualbox, Docker, . . . ) or even set up a real machine through network boot.

2. the **setup** phase can now involve the “in” context, that is inside the new OS, to install and configure the new system.

[13] https://github.com/oar-team/kameleon
[14] https://github.com/oar-team/kameleon-recipes

3. the **export** phase is extracting the OS image in the wanted format, and shut down the “in” context.

As explained in the Section 6.1, the reproducibility with variation requires the environment reconstructability. Kameleon is providing that, as long as all the external sources of information are available (source code, binaries, etc.). To avoid this issue an additional mechanism is implemented is provided by Kameleon, the artifact caching during all building process. This caching mechanism enables to reconstruct the images offline by caching all the artefacts downloaded from external sources during the first building process. This system has limitations: it currently only works for HTTP connections, and rebuilding from the cache but with modifications is not trivial. Overcoming those limitations is in the roadmap of Kameleon.

Kameleon is, to the best of our knowledge, the only tool that exposes this notion of execution contexts, making it very generic for any kind of complex remote execution work. This concept could be applied to Continuous Integration (CI) tools that have very similar needs but lacks flexibility.

**Usage during this thesis** The experience gained from the development of Kameleon and especially Kameleon recipes was very important to understand package managers and virtualization technologies discuss in the Chapter 3.

Also, the reflection on reproducibility was derived from the experience of building reproducible experiments with Kameleon.

Kameleon was directly used to generate reproducible OS images for all the real conditions experiments of this thesis: The Batsim validation process [Dut+16] and the evaluation of Bebida 4.5.2.

### 6.4.2 Batsim

Batsim [Dut+16] is an infrastructure simulator made for scheduling experiment and platform dimensioning. The purpose of Batsim is to provide a fair comparison between scheduling algorithms on top a realistic infrastructure model. It has the capacity to interact with a scheduler coded in any language, through a simple text-based network protocol. At the time of this writing, more than 5 schedulers have been implemented, with for some of them tens of scheduling policies. The infrastructure simulation is done through a distributed system framework called Simgrid [Cas+14]. It is based on platform definition files that let users describe arbitrary infrastructure depending on their needs. This allows the modeling of the computing infrastructure resources and of the network. Simgrid features different level of application models that are partly validated experimentally [FC07]. These various application models are exposed by Batsim to the users, allowing them to choose the appropriate trade-off between realism and pace of simulation. The workload models proposed by Batsim are a composition of application profiles associated with their time of arrival, and resources requests. The simulator is also capable to model data movements across the simulated platform as described in a previous Chapter (5). The accuracy of Batsim regarding scheduling metrics was evaluated experimentally [Dut+16].

Batsim is a powerful tool for HPC scientists and operators It let them test and compare different scheduling policies, and their impact on different platforms, depending on a given workload. This task cannot be done in real conditions because it is not possible to monopolize an HPC platform long enough to perform this kind of study correctly. Another approach is to emulate this kind of platform with a resource folding, but emulation is limited by the fact that it relies on real hardware that prevents to reproduce real conditions [GNQ13].

The most similar approach is ALEA [KTP16] which is based on GridSim [Sul+08]. ALEA is coded in Java and only Java scheduler can be added to it using, thus limiting the comparison between scheduling algorithms. However, ALEA has a very interesting feature, the user's modeling based on the work of Feitelson around workload modeling [Fei15a]. Adding a user model is one of the future works of Batsim.

The CODES/ROSS [Wan+18] simulator is an infrastructure simulator developed in Argonne National Laboratory. This simulator's approach is similar to Simgrid, but focus on network simulation precision. It is based on a very low-level network abstraction (packet level), whereas Simgrid provides a flow level model that is much less costly to compute and yet accurate in most cases [FC07].

One strong point of Batsim is its validation regarding scheduling metrics. This was made possible thanks to the design of Batsim that allow plugging any scheduler on the simulator. Indeed, a scheduler used in a real RJMS (OAR3) called Kamelot was plugged on top of Batsim, using a thin adaptation layer, the Batsim Adaptor for OAR (Bataar), so we can experiment with the exact same scheduling algorithm in simulation and in real conditions. The Figure 6.3 depicts this process.

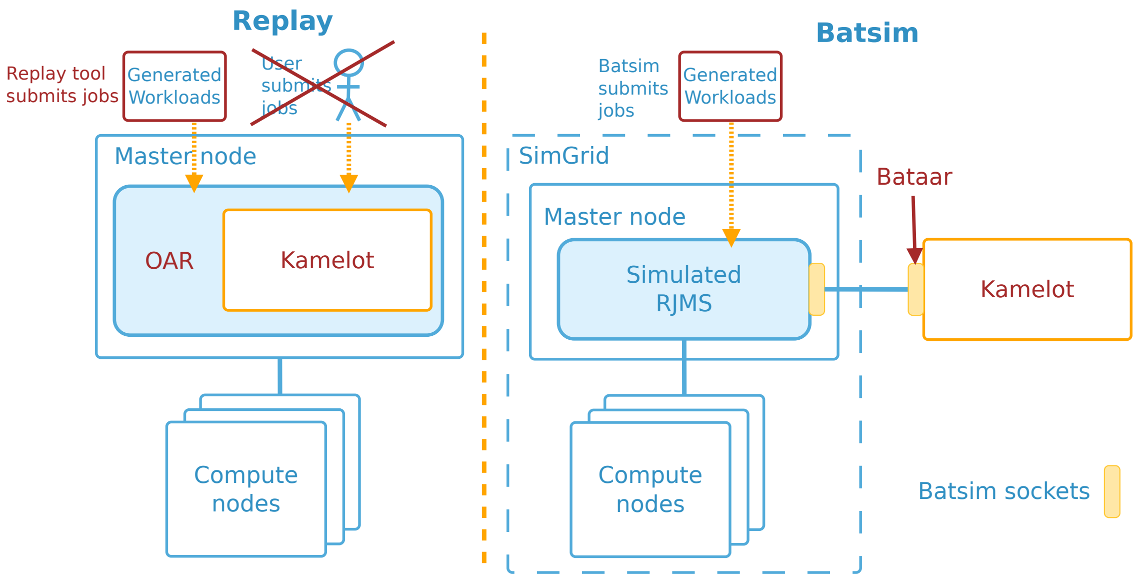


Figure 6.3.: Batsim validation process using a real world Resource and Job Management System scheduler.

The evaluation results indicate that the difference between simulation and real execution is in the same range as the difference between two real executions for standard scheduling metrics:

- Makespan a.k.a. Cmax:

$$\max_i(FinishT_i)$$

- Mean bounded slowdown

$$\frac{1}{nbJobs}\sum_i \max\left(\frac{FinishT_i - SubmitT_i}{\max_i(K, FinishT_i - StartT_i)}, 1\right)$$

The Figure 6.4 shows the differences between two real executions and a simulated execution with respect to two of these metrics, namely the Mean Bounded Slowdown and the Makespan.

Another strength of Batsim is its ecosystem. Batsim is heavily tested in a Continuous Integration (CI) tool for every modification with more than 400 tests. The Batexpe project[15] is made for easing experiment with Batsim, giving a simple descriptive

[15]https://framagit.org/batsim/batexpe

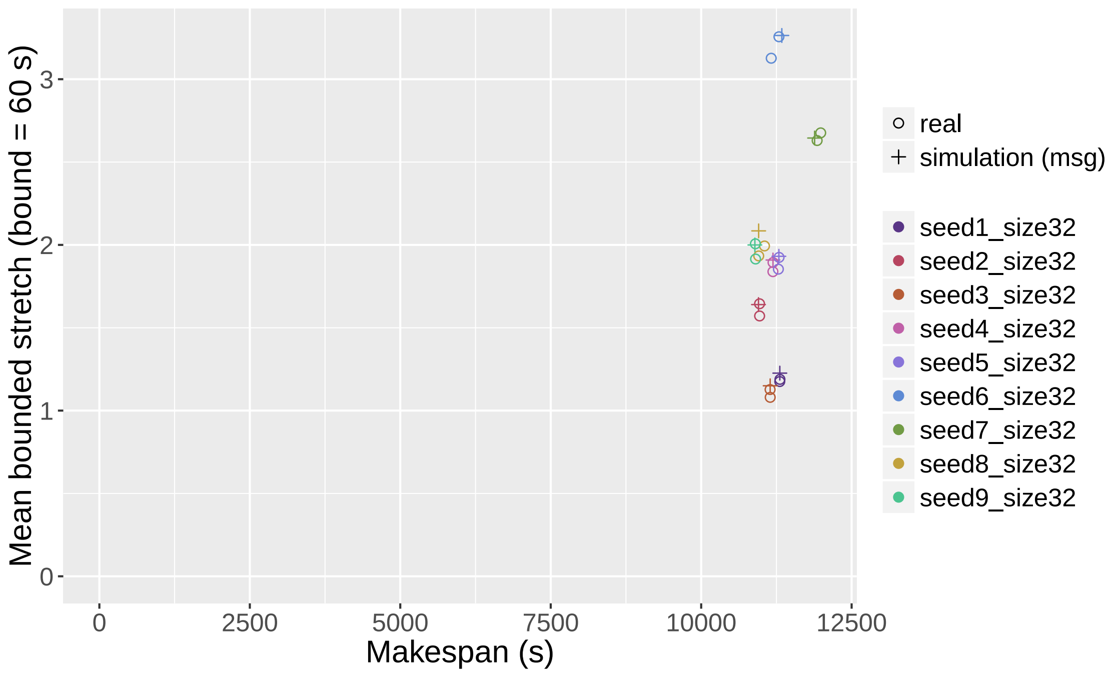


**Figure 6.4.:** Comparison between two real scheduling executions and a simulated one done with Batsim, with respect to two metrics, (Mean Bounded Slowdown and Makespan) over 9 different workloads.

way to manage the lifecycle of the two processes required to run a simulation: one for the scheduler, and one for Batsim itself.

Batsim is for now limited to simulate only centralized RJMS, but, it represents most of the actual use cases, and emulating distributed scheduling on top of the Batsim protocol is already tested.

In the Batsim roadmap, we want to add more application models for HPC jobs. This notably includes the simulation of regular I/O patterns.

**Usage during this thesis** During this thesis, we have published a paper on Batsim itself, that explains the underlying principles, and proposes an experimental validation [Dut+16].

The work on Bebida 4.1, notably the generation of the workloads used in the paper have been done thanks to Batsim.

The Chapter 5 is entirely based on simulations on top of Batsim. This study leverages the fact that having a text-based protocol gives great flexibility because the Bebida prolog/epilog mechanism was implemented in a simple message broker without modifying neither Batsim nor the two schedulers it interacts with. Batsim capacities were also extended with storage resources to be able to model parallel file systems and distributed file systems.

We think that one of the Batsim goal, which is to improve collaboration among the scientific community, is already fulfilled: We already have 7 contributors from our team, and 5 external contributors from Brazil, Argentine, Germany, Poland, and the U.S.A.

### 6.4.3 Evalys

Evalys is a resource and job scheduling performance evaluation framework. It is designed to load, process, and display information about scheduling traces. It accepts two formats: The Standard Workload Format (SWF) as defined by Feitelson [Cha+99] and documented on a website [Fei18]. This format is the standard that was chosen for the Parallel Workload Archive [Fei17]. This archive contains 38 cluster traces from 1993 to 2015. The other format that is supported by Evalys is the so-called "JobSet" format corresponds to the Batsim output format.

It is very similar to the SWF format but it contains additional information such as the set of resources that were used for each job, i.e., the actual job location on the cluster. Evalys also allows for visualizing energy consumption and machines' power states.

Figure 6.5 is a detailed visualization of a JobSet generated by a simple call to the "plot()" function. The four graphs share the same X axis, which is the time in seconds. The Y axis of all those graphs is a number of resources, here a machine (a.k.a. compute nodes).

The **utilization** graph shows the number of resources used by a job over time. With the green line being the maximum number of resources. Note that sometimes the utilization goes upper the green line but it is just a visualization artifact due to events seen as simultaneous due to rounding.

The **queue** graph is the sum of all the resources requested by all the job in the queue over time. It represents the load of the cluster.

The **job details** visualization (based on an original idea of David Glesser) is one of the more interesting. Each job is represented by three elements joined by lines that represent the three events in the life of a job. The first element is a blue dot that corresponds to the submission time, the green triangle is the starting time and the red bar the finish time. Those three elements are not aligned in the Y axis to improve the readability by avoiding superposition of elements. That's why each element are separated by the total number of resources (here 128) and they are all translated by the number of requested resources. The bigger the job is (in number of resources), the upper the 3 elements are translated. It is worth noticing that we can observe job's waiting time by observing the slope of the blue: if a job doesn't wait, its associated blue line is vertical, and the more a job is waiting the more the blue line became horizontal. The same observation can be done for the jobs runtime and the green line.

The **Gantt chart** visualization is showing where and when each job has run during the simulation. The Y axis is an arbitrary ordering based on the computing resources' IDs. It has to be correlated with the network topology given to the simulator with a mapping between nodes IDs and nodes position on the topology to fully understand allocation policies.

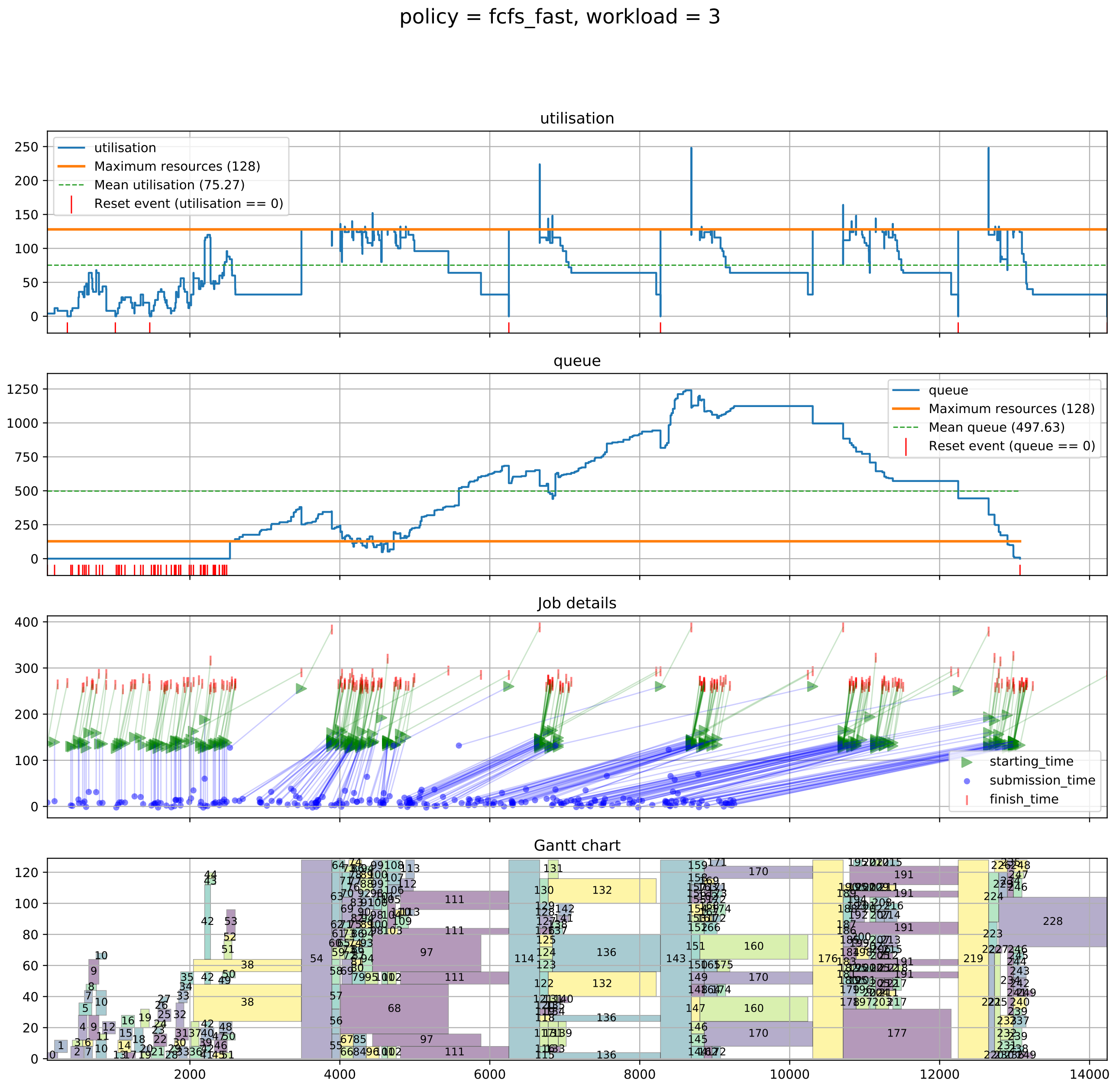


Figure 6.5.: Batsim simulation output visualization using Evalys. It permits to the Batsim users to rapidly check on the cluster utilization, the queue load in number of resources asked but not attributed yet, the details for each job events, and the jobs allocation. More details in this section.

**Usage during this thesis** Evalys was used, and still is, for visualizing and exploring SWF files and Batsim output files. It allows us, the Datamove team and all the users of Batsim, to collaborate on complex visualization elaboration and common operation like computing mean utilization and queue or extracting some part of a workload regarding utilization conditions.

### 6.4.4 Kapack

Kapack is a Nix packages collection maintained by actual and former members of the Datamove team. It contains a set of packages used by the team for multiple experiments, plus some utilities[16].

**Usage during this thesis** Kapack provides us a good way to collaborate on the packaging of complex software stack like the ones requires by Batsim. For example, due to the fast development of Simgrid: Three projects from the team are using three different versions of Simgrid as Batsim dependency. Packaging this with Nix allows us to provide a one-liner to install each version, but also development environments. All this was done with a few lines of Nix DSL thanks to Nix composability.

### 6.4.5 Grid'5000 Big Data experiment tools

This project is based on the extraction of the reusable part of the code I have made for the BeBiDa real conditions experiments. It is composed of customizable Kameleon recipes and a python library based on Execo. This library contains all the needed procedures to set up the whole HPC and Big Data stack, e.g. the SSH configuration, the launch of all kind of services, the formatting HDFS, the dataset ingestion in HDFS, and so on[17].

It was used as a basis for a Grid'5000 Big Data tutorial that was presented to the Grid'5000 community at the Grid'5000-FIT school by Pierre Neyron in April 2018[18]. It was also used by two Master interns to set up their experiments on Grid'5000.

[16]https://github.com/oar-team/kapack
[17]https://gitlab.inria.fr/evalys/big-data-hpc-g5k-expe-tools
[18]https://www.grid5000.fr/mediawiki/index.php/BigData_hands-on_tutorial

## 6.5 Conclusion

Reproducibility is a major tool to assess experimental science. But trying to provide reproducible experiments takes time, and it is poorly considered in software-based science in general, and in computer science in particular.

But, taking the time to produce a reproducible experiment is worthwhile. It brings better confidence in the results and allows to lay trusted foundations for further research. This is valid for external reviewers, for (future) team members, but it is even more important for the original author: embracing high standards of methodology and engineering, produces reusable bricks that will hopefully be very valuable to continue research work.

Moreover, the complexity of scientific research has become so high that collaboration on tools' development starts to be mandatory. Indeed, a Ph.D. student in a few years have not enough time to rebuild experimental tools that were previously done by is predecessor, but lost by lack of knowledge sharing.

In this thesis, we have tried to define and apply good practices, and we manage to produce reproducible experiments using suitable methods and tools. Part of the real conditions experiments were reproduced with variation by myself, and some software environments were already reused by other team members.

We have also produced software tools with sustainability in mind, and it pays off because most of them are already used by team members but also by external researchers.

To conclude, reproducibility is a long-term target with the positive side-effect to lead the scientific community to produce stronger results, to share the knowledge more efficiently, and finally to cooperate more widely.

# Conclusion 7

In this thesis, we have explored different facets of the Big Data and HPC convergence, especially from the infrastructure point of view. The following lists our contributions in this field.

## 7.1 Contributions

In Chapter 2 we have clarified what really is Big Data and HPC through each field's history, while providing a definition of each element that characterized them, beyond the buzz words. By comparing them and exposing their differences and similarities, we propose a state-of-the-art of the Big Data and HPC convergence, while highlighting the context and the challenges of this rapprochement.

One of the challenges that have to be tackled is software provisioning. Indeed, in the convergence context, a common way to deploy users' software stack that can manage HPC, Big Data and hybrid applications is required. Chapter 3 provides a comprehensive survey on software provisioning techniques applicable to HPC systems, with a focus on data-intensive workload needs. It contains a definition of the different stages of the software provisioning as well as their associated requirements. Then, all the main approaches are reviewed with respect to these requirements.

We conclude that while there is a promise on container-based solutions, they failed to fit all the users' needs. One of the issues is the unwanted isolation that hinders access to the HPC platform hardware and the portability. The use of functional package managers seems to be a better match to the requirements of a hybrid software stack, with a more integrated, transparent, and customizable way of packaging. Table 3.1 provides a summary of the different approaches and how they implement the requirements.

Another aspect of the convergence is the resource management, and how to make computation and data-intensive workloads co-exist on the same platform. Chapter 4 discusses existing approaches and propose a new technique based on Resource and Job Management System's (RJMS) collaboration.

In classic HPC workloads, the rigidity of jobs tends to create 'holes' (of un-utilized resources) in the schedule: we can use those idle resources as a dynamic pool for the Big Data workloads. We propose a new idea called BeBiDa for Best-Effort Big Data, solely based on RJMS configuration. It makes HPC and Big Data systems communicate through a simple prolog/epilog mechanism, which leverages the built-in resilience of Big Data frameworks while minimizing the disturbance on the HPC workloads.

We present the first study of this approach, using the production RJMS middleware OAR and Hadoop YARN, from the HPC and Big Data ecosystems respectively. Note that BeBiDa can be adapted to any HPC RJMS that feature prolog/epilog and any Big Data RJMS that is capable to manage a dynamic set of resources. Our new technique is evaluated with real experiments upon the Grid5000 platform. Our experiments validate our assumptions and show promising results. The system is capable of running an HPC workload with a cluster utilization of 70%, featuring a Big Data workload that fills the schedule holes to reach a full 100% utilization. We observe a penalty on the mean waiting time for HPC jobs of less than 17% and a Big Data effectiveness of more than 68% on average (see Section 4.5.2). Our approach is based on configuration and needs only 50 lines of additional code to run while alternative approaches use 100x to 1000x more lines of code.

Experimenting on converged Big Data and HPC infrastructure like the one proposed by the BeBiDa approach requires the use of simulation. Nevertheless, existing infrastructure simulators are mostly tight to a specific tool, and not capable to simulate the I/O behavior of the applications. The first problem is solved by using our infrastructure simulator Batsim which is scheduler agnostic. To deal with the second one, we chose to incorporate an I/O model within Batsim.

In Chapter 5 we define a new I/O aware platform simulation approach, based on Batsim and his underlying simulation framework SimGrid. We expose a new Big Data application model generated from Spark application traces, and a generic I/O transfer model capable of reflecting the behavior of centralized and distributed file systems.

Experimental results show that thanks to Batsim, we can reproduce BeBiDa scheduling scheme and Big Data application performance metrics. In addition, we managed to study file system trade-offs between centralized and distributed file systems: we concluded that both file systems are equivalent for applications running alone but the network access link to the centralized file system becomes an important point of contention when confronted with a larger data-intensive workload.

Finally, all the work that was done during this thesis was done with reproducibility in mind, and reproducibility itself is one of our research subjects. Indeed, Scientific knowledge production is based on experiments that rely more and more on software, and while the science progresses the daily used software tools are becoming numerous and complex. This complexity exacerbates the difficulty to reproduce scientific results based on software without a proper methodology.

In Chapter 6 we expose observations and feedbacks from our multiple tries to achieve reproducibility in different contexts. Indeed, during this thesis, we have done experiments in real conditions and in simulation. Based on this experience, we are proposing a new way of seeing reproducibility which takes into account the software development workflow. Also, we define the experimental data analysis workflow and how it should be integrated into the reproducibility mindset. Finally, we provide a list of good practices for reproducible experiments, and we conclude on how it was implemented in this thesis with a detailed description of the tools that were developed and used.

We conclude that reproducibility is not just a matter of making repeatable science, it also improves confidence in results by allowing others to inspect and review the work that was done. Moreover, it enables internal and external researchers to reuse tools, contribute to them, and collaborate with each other.

To summarize the contributions of this thesis, we have:

- Proposed a state-of-the-art on the Big Data and HPC convergence.
- Proposed a survey on the software provisioning on HPC with a focus on the data-intensive stack.
- Defined and evaluated an original and simple approach of resource management collaboration called BeBiDa.
- Developed and evaluated an infrastructure simulator called Batsim, and made it I/O aware.

- Used this simulator to reproduce and extend the results obtained in real conditions for BeBiDa.

- Used this simulator to show that we can experiment on I/O aware platform dimensioning. We showed for example that even on a small platform a full Big Data workload requires a DFS instead of a PFS in order to avoid network congestion.

- Defined a new way of seeing reproducibility that includes software environments and the scientific software development workflow.

## 7.2 Perspectives

One of my Thesis Director, Olivier Richard, told me once that, an exploratory thesis like this one, is successful if it leads to three new thesis subjects. The following subsections are three different scientific paths that can hopefully be used as a basis for future (thesis) works.

### 7.2.1 Improve HPC and Big Data systems collaboration and resilience

Because Big Data and HPC systems are different, the convergence of both worlds will not happen through a merge of all the tools used in the community but through a better collaboration of the tools specialized for both workloads.

Moreover, in the path to Exascale, the HPC community is seeking better fault-tolerance to hardware failure which requires for the different software levels to collaborate.

Improving collaboration can be achieved through different paths, but it always requires communication. This communication can be either explicit, where both tools know each other, or blind, where each tool exposes information about its behavior for others to take smart decisions accordingly i.e., observability.

Explicit collaboration through a common API is already used in the Cloud and in the Big Data community e.g., the Mesos framework API is used outside Mesos as a universal API for resource request [Uni19]. In the Grid and HPC world, the Distributed Resource Management Application API (DRMAA) [Tro+07] is the standard API to request resources to an RJMS, but it was limited and the second version developed in 2012 is still not widely adopted.

The BeBiDa mechanism proposed in Chapter 4 illustrates the collaboration capabilities of complex systems, but in order to improve them we need observability of both systems. There are already some examples of the benefits of observability API at different levels of the infrastructure: for example, the OpenMP Tools (OMPT) interface, that exposes idle state of the OpenMP computation, can be used to improve MPI asynchronous communication [Ser+18].

We advocate for both types of communication to be generalized at every level of the HPC and Big Data infrastructures. This will enable to unleash collaboration possibilities around better scheduling (jobs, I/O, Network bandwidth), improve the quality of service, provide file system optimizations, and a lot more yet to be imagined.

### 7.2.2 Improve workload modeling for a better HPC and Big Data system simulation

The main input of a cluster infrastructure simulation is the workload. But, even if this subject was studied in both HPC and Big Data community (see Section 2.2), there is no advanced workload model that permits to experiment all the variety of user's behaviors on top of hybrid platforms. Most of the current studies in the scheduling field are done over real cluster utilization traces that are incomplete and too specific to be representative. This leads us to these questions: Are there representative workloads? How to find the applicability of our approach without experimenting on them?

We think that statistical modeling approach of HPC and Big Data workload is only the first step toward a more advanced platform workload modeling. Furthermore, basing our scientific studies on workload from the past (sometimes decades old) instead of trying to model the workload of the future, gives to experimental results very low credits and make them inapplicable in real life.

Also, regarding the sensitive nature of parallel jobs scheduling, where a small decision change can have a huge impact in the future, the robustness of our scheduling algorithm should be tested on large sets of workload. Moreover, when questioning the HPC users [RSR15], they claim that they react to the cluster state by going to another cluster when one is too utilized, and also that they are using intermediate tools that make bags of jobs. These statements make most of the statistical model ill-suited because they are based on the assumption that jobs' submissions are independent and cannot dynamically react to the cluster state.

One way to model the workloads properly is to take a step back and to model cluster users. Feiltelson already proposed a user model based on this assumption [Fei16]. The generative model of workloads based on users' behavior model seems to be promising because it permits creating dynamic workloads by modeling the way users respond to the platform state.

We don't have a clear idea of what the future workloads will be. However, we already have an idea of how current users are interacting with the platform, and we can start by properly modeling this. Then, it would be easier to extrapolate on the changes in their behavior and test the impact of these changes in the simulation.

But simulating workloads is not only modeling how users are submitting jobs to the platform but also modeling the jobs themselves. In this thesis, we have started to model jobs at the platform level (see Chapter 5), but we need to continue this work by improving the Big Data model, and modeling HPC jobs with I/O.

Without a proper simulation of these systems, improving the way we are designing and using this kind of system will always be empirical. Trying to catch up with the current platforms' usage is very important for the simulation experiment research in order to better design current and future platforms yet to come.

### 7.2.3 Towards a reproducible computing infrastructure

Developing standard methodologies and appropriate tools for reproducibility is a good way to improve trust in software-based science and increase collaboration, but it can also be seen as a scientific subject in itself.

The issue of producing reproducible software is already in progress in the Linux distributions community [com19c], but what about the whole infrastructure. How to reproduce a whole cluster setup? The issue is not just about software anymore, but also about resources state. Going from scratch to a fully functional infrastructure involves multiple elements: system software provisioning, file systems, the scheduling and resource management system, and so on.

It also raises the question of state management. An IT infrastructure is a distributed moving target that evolves over time. How can we track, checkpoint, and restore the whole infrastructure state: the most obvious point to tackle is the application data, from the database to the logging system, going through all privates and shared file systems. But, there is also the resources state, i.e., the configuration from the firmware to the OS, and the services state, i.e., which services are running and where. This becomes even more complex if there are distributed systems involved.

As exposed in Section 3.6.1, testbeds for computer science like Emulab [Whi+02], Chameleon [MCY15], and Grid5000 [Bal+13], are providing the root requirements for reproducible infrastructure, because they feature ways to save and restore OS environment and network configuration.

Other aspects of the problem like service and state management are already explored by the Nix community with NixOps [com19b] and Disnix [com19a] projects. Another approach is to use configuration management tools like Terraform [Has19], Puppet [Pup19], Ansible [Hat19a], or SaltStack [Hat19b]. However, the lack of clearly defined abstractions, prevent them to manage the state issue properly.

Having this level of definition for infrastructures would be very beneficial for the industrial and scientific communities, but it requires the right level of abstraction along with development of the associated tools.

# Appendix A

## A.1 BeBiDa full implementation

### A.1.1 prolog

**Prolog script:**

```bash
#!/bin/bash

exec >> /tmp/oar_prologue_$1 2>&1
set -x

export YARN_HOST='localhost'

# Get nodes adresses from oarstat
nodes=`oarstat -Yj $1 | \
  python -c 'import yaml,sys;print(yaml.safe_load(sys.stdin)['$1']["assigned_network_address"])
  | tr -d "'[],"`

echo "nodes=$nodes"

ssh -tt $YARN_HOST -- "su - hadoop -c '\$HADOOP_CONF_DIR/hadoop_nm_stop.sh \"$nodes\"'"

# Wait for the NodeManager to finish
for node in $nodes; do
  ssh -tt $node -- "bash -s" < $HADOOP_CONF_DIR/wait_for_nm.sh &
done

wait
exit 0
```

**Stop Hadoop node manager script:**

```bash
#!/bin/bash

# This script must bu run as hadoop user
nodes=$1

# Stop NodeManagers
for node in $nodes
do
  echo $node >> $HADOOP_CONF_DIR/yarn.exclude
done

# Update yarn nodemanager state
$HADOOP_HOME/bin/yarn rmadmin -refreshNodes
```

**Wait for the node manager to be stopped:**

```bash
NM_PID=$(pgrep -u hadoop -f org.apache.hadoop.yarn.server.nodemanager.NodeManager)
if [ -z $NM_PID ]
then
  echo "WARNING: The node manager is not running (already stopped)"
else
  echo "Waiting for the node manager to be stopped (PID=$NM_PID)"
  while [ -e /proc/$NM_PID ]; do sleep 0.1; done
fi

exit 0
```

### A.1.2 epilog

**Epilog script:**

```bash
#!/bin/bash

exec >> /tmp/oar_epilogue_$1 2>&1
set -x

export YARN_HOST='localhost'

# Get nodes adresses from oarstat
```

```bash
nodes=`oarstat -Yj $1 | \
  python -c 'import yaml,sys;print(yaml.safe_load(sys.stdin)['$1']["assigned_network_address"])
  | tr -d "'[],"`

ssh -tt $YARN_HOST -- "su - hadoop -c '\$HADOOP_CONF_DIR/hadoop_nm_start.sh \"$nodes\"'"

# Start yarn nodemanager
for node in $nodes; do
  ssh -tt $node -- \
    'su - hadoop -c "$HADOOP_HOME/sbin/yarn-daemon.sh start nodemanager"' &
done

wait

exit 0
```

**Start Hadoop node manager script:**

```bash
#!/bin/bash

# This script must bu run as hadoop user
nodes=$1

# Remove node from excluded
for node in $nodes; do
  sed -i /$node/d $HADOOP_CONF_DIR/yarn.exclude
done

# Update yarn nodemanager state
$HADOOP_HOME/bin/yarn rmadmin -refreshNodes
```

# Bibliography

[ABE18] Guillaume Aupy, Olivier Beaumont, and Lionel Eyraud-Dubois. *Sizing and Partitioning Strategies for Burst-Buffers to Reduce IO Contention*. Inria, 2018 (cit. on p. 29).

[ACP95] T. E. Anderson, D. E. Culler, and D. Patterson. „A case for NOW (Networks of Workstations)". In: *IEEE Micro* 15.1 (Feb. 1995), pp. 54–64 (cit. on p. 7).

[Ala+11] Sadaf R. Alam, Hussein N. El-Harake, Kristopher Howard, Neil Stringfellow, and Fabio Verzelloni. „Parallel I/O and the Metadata Wall". In: *Proceedings of the Sixth Workshop on Parallel Data Storage*. PDSW '11. Seattle, Washington, USA: ACM, 2011, pp. 13–18 (cit. on p. 29).

[Amv+18] George Amvrosiadis, Jun Woo Park, Gregory R. Ganger, et al. „On the diversity of cluster workloads and its impact on research results". In: *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, 2018, pp. 533–546 (cit. on p. 11).

[Ana+09] Rajagopal Ananthanarayanan, Karan Gupta, Prashant Pandey, et al. „Cloud Analytics: Do We Really Need to Reinvent the Storage Stack?" In: *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*. HotCloud'09. San Diego, California: USENIX Association, 2009 (cit. on p. 29).

[And+03] Nazareno Andrade, Walfredo Cirne, Francisco Vilar Brasileiro, and Paulo Roisenberg. „OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing". In: *Job Scheduling Strategies for Parallel Processing, 9th International Workshop, JSSPP 2003, Seattle, WA, USA, June 24, 2003, Revised Papers*. 2003, pp. 61–86 (cit. on p. 69).

[And04] D. P. Anderson. „BOINC: a system for public-resource computing and storage". In: *Fifth IEEE/ACM International Workshop on Grid Computing*. 2004, pp. 4–10 (cit. on p. 69).

[Asc+18] M Asch, T Moore, R Badia, et al. „Big data and extreme-scale computing: Pathways to Convergence-Toward a shaping strategy for a future software and data ecosystem for scientific inquiry". In: *The International Journal of High Performance Computing Applications* 32.4 (2018), pp. 435–479. eprint: https://doi.org/10.1177/1094342018778123 (cit. on pp. 14, 15, 51).

[Ave+16] Guilherme Avelino, Leonardo Teixeira Passos, André C. Hora, and Marco Tulio Valente. „A Novel Approach for Estimating Truck Factors“. In: *CoRR* abs/1604.06766 (2016). arXiv: 1604.06766 (cit. on p. 128).

[Aza17] Abdulrahman Azab. „Enabling docker containers for high-performance and many-task computing“. In: *Cloud Engineering (IC2E), 2017 IEEE International Conference on*. IEEE. 2017, pp. 279–285 (cit. on p. 50).

[Bal+13] Daniel Balouek, Alexandra Carpen Amarie, Ghislain Charrier, et al. „Adding Virtualization Capabilities to the Grid'5000 Testbed“. In: *Cloud Computing and Services Science*. Ed. by Ivan I. Ivanov, Marten van Sinderen, Frank Leymann, and Tony Shan. Vol. 367. Communications in Computer and Information Science. Springer International Publishing, 2013, pp. 3–20 (cit. on pp. 52, 77, 148, 168).

[Bhi+17] Janki Bhimani, Zhengyu Yang, Miriam Leeser, and Ningfang Mi. „Accelerating big data applications using lightweight virtualization framework on enterprise cloud“. In: *High Performance Extreme Computing Conference (HPEC), 2017 IEEE*. IEEE. 2017, pp. 1–7 (cit. on p. 33).

[Bir+15] M. S. Birrittella, M. Debbage, R. Huggahalli, et al. „Intel® Omni-path Architecture: Enabling Scalable, High Performance Fabrics“. In: *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*. Aug. 2015, pp. 1–9 (cit. on p. 22).

[Boe15] Carl Boettiger. „An introduction to Docker for reproducible research“. In: *ACM SIGOPS Operating Systems Review* 49.1 (2015), pp. 71–79 (cit. on p. 133).

[Buz+80] B.L. Buzbee, W.J. Worlton, G. Michael, and G. Rodrigue. „DOE research in utilization of high-performance computers“. In: (Dec. 1980) (cit. on p. 6).

[Bze+17] Bruno Bzeznik, Oliver Henriot, Valentin Reis, Olivier Richard, and Laure Tavard. „Nix as HPC package management system“. In: *Proceedings of the Fourth International Workshop on HPC User Support Tools*. ACM. 2017, p. 4 (cit. on p. 58).

[CAK12] Yanpei Chen, Sara Alspaugh, and Randy Katz. „Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads“. In: *Proceedings of the VLDB Endowment* 5.12 (2012), pp. 1802–1813 (cit. on pp. 12, 13).

[Cap+05] Nicolas Capit, Georges Da Costa, Yiannis Georgiou, et al. „A batch scheduler with high level components“. In: *CCGrid 2005*. 2005 (cit. on p. 68).

[Car+15] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, et al. „Apache Flink™: Stream and Batch Processing in a Single Engine“. In: *IEEE Data Eng. Bull.* 38 (2015), pp. 28–38 (cit. on p. 68).

[Cas+14] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. „Versatile, scalable, and accurate simulation of distributed applications and platforms". In: *Journal of Parallel and Distributed Computing* 74.10 (Oct. 2014), pp. 2899–2917 (cit. on pp. 91, 93, 148, 154).

[Cas+18] Henri Casanova, Suraj Pandey, James Oeth, et al. „WRENCH: A Framework for Simulating Workflow Management Systems". In: *WORKS 2018 - 13th Workshop on Workflows in Support of Large-Scale Science*. Dallas, United States, Nov. 2018, pp. 1–12 (cit. on p. 93).

[CE97] Michael Cox and David Ellsworth. „Application-controlled Demand Paging for Out-of-core Visualization". In: *Proceedings of the 8th Conference on Visualization '97*. VIS '97. Phoenix, Arizona, USA: IEEE Computer Society Press, 1997, 235–ff. (Cit. on p. 8).

[Cha+16] Nicholas Chaimov, Allen Malony, Shane Canon, et al. „Scaling Spark on HPC systems". In: *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*. ACM. 2016, pp. 97–110 (cit. on p. 18).

[Cha+99] Steve J. Chapin, Walfredo Cirne, Dror G. Feitelson, et al. „Benchmarks and Standards for the Evaluation of Parallel Job Schedulers". In: *Job Scheduling Strategies for Parallel Processing*. Ed. by Dror G. Feitelson and Larry Rudolph. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 67–90 (cit. on p. 157).

[Chi+16] Fernando Chirigati, Rémi Rampin, Dennis Shasha, and Juliana Freire. „ReproZip: Computational Reproducibility With Ease". In: *Proceedings of the 2016 International Conference on Management of Data*. SIGMOD '16. San Francisco, California, USA: ACM, 2016, pp. 2085–2088 (cit. on p. 133).

[Cou13] Ludovic Courtès. „Functional Package Management with Guix". In: *CoRR* abs/1305.4584 (2013). arXiv: `1305.4584` (cit. on p. 57).

[CPW15] Christian Collberg, Todd Proebsting, and Alex M Warren. „Repeatability and benefaction in computer systems research". In: *University of Arizona TR 14* 4 (2015) (cit. on pp. 126, 140).

[CW15a] Ludovic Courtès and Ricardo Wurmus. „Reproducible and User-Controlled Software Environments in HPC with Guix". In: *Euro-Par 2015: Parallel Processing Workshops*. Ed. by Sascha Hunold, Alexandru Costan, Domingo Giménez, et al. Cham: Springer International Publishing, 2015, pp. 579–591 (cit. on p. 131).

[CW15b] Ludovic Courtès and Ricardo Wurmus. „Reproducible and user-controlled software environments in HPC with Guix". In: *European Conference on Parallel Processing*. Springer. 2015, pp. 579–591 (cit. on p. 58).

[DDS15a] Adrien Devresse, Fabien Delalondre, and Felix Schürmann. „Nix Based Fully Automated Workflows and Ecosystem to Guarantee Scientific Result Reproducibility Across Software Environments and Systems“. In: *Proceedings of the 3rd International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering*. SE-HPCCSE '15. Austin, Texas: ACM, 2015, pp. 25–31 (cit. on p. 58).

[DDS15b] Adrien Devresse, Fabien Delalondre, and Felix Schürmann. „Nix based fully automated workflows and ecosystem to guarantee scientific result reproducibility across software environments and systems“. In: *Proceedings of the 3rd International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering*. ACM. 2015, pp. 25–31 (cit. on p. 133).

[DDV+04] Eelco Dolstra, Merijn De Jonge, Eelco Visser, et al. „Nix: A Safe and Policy-Free System for Software Deployment.“ In: *LISA*. Vol. 4. 2004, pp. 79–92 (cit. on p. 56).

[Deg+17] Augustin Degomme, Arnaud Legrand, Georges Markomanolis, et al. „Simulating MPI applications: the SMPI approach“. In: *IEEE Transactions on Parallel and Distributed Systems* 28.8 (Aug. 2017), p. 14 (cit. on pp. 92, 94).

[Der+15] S. Derradji, T. Palfer-Sollier, J. Panziera, A. Poudes, and F. W. Atos. „The BXI Interconnect Architecture“. In: *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*. Aug. 2015, pp. 18–25 (cit. on p. 22).

[DG04] Jeffrey Dean and Sanjay Ghemawat. „MapReduce: Simplified Data Processing on Large Clusters“. In: *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*. OSDI'04. San Francisco, CA: USENIX Association, 2004, pp. 10–10 (cit. on p. 8).

[DG08] Jeffrey Dean and Sanjay Ghemawat. „MapReduce: simplified data processing on large clusters“. In: *Communications of the ACM* 51.1 (2008), pp. 107–113 (cit. on pp. 16, 68).

[DKC12] S. Di, D. Kondo, and W. Cirne. „Characterization and Comparison of Cloud versus Grid Workloads“. In: *2012 IEEE International Conference on Cluster Computing*. Sept. 2012, pp. 230–238 (cit. on p. 11).

[Don+79] J.J. Dongarra, J.R. Bunch, C.B. Moler, and G.W. Stewart. *LINPACK Users' Guide*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, 1979 (cit. on p. 6).

[Dün+16] Celestine Dünner, Thomas P. Parnell, Kubilay Atasu, Manolis Sifalakis, and Haralampos Pozidis. „High-Performance Distributed Machine Learning using Apache SPARK“. In: *CoRR* abs/1612.01437 (2016). arXiv: `1612.01437` (cit. on p. 19).

[Dut+16] Pierre-François Dutot, Michael Mercier, Millian Poquet, and Olivier Richard. „Batsim: a Realistic Language-Independent Resources and Jobs Management Systems Simulator“. In: *20th Workshop on Job Scheduling Strategies for Parallel Processing*. Chicago, United States, May 2016 (cit. on pp. 79, 91, 148, 153, 154, 157).

[Fad+12] Z. Fadika, M. Govindaraju, R. Canon, and L. Ramakrishnan. „Evaluating Hadoop for Data-Intensive Scientific Operations“. In: *2012 IEEE Fifth International Conference on Cloud Computing(CLOUD)*. Vol. 00. June 2012, pp. 67–74 (cit. on p. 30).

[FC07] Kayo Fujiwara and Henri Casanova. „Speed and accuracy of network simulation in the simgrid framework“. In: *Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 2007, p. 12 (cit. on pp. 94, 154).

[Fei15a] Dror Feitelson. *Workload Modeling for Computer Systems Performance Evaluation*. Cambridge: Cambridge University Press, 2015 (cit. on pp. 11, 79, 154).

[Fei15b] Dror G Feitelson. „From repeatability to reproducibility and corroboration“. In: *ACM SIGOPS Operating Systems Review* 49.1 (2015), pp. 3–11 (cit. on p. 128).

[Fei16] Dror G. Feitelson. „Resampling with Feedback – A New Paradigm of Using Workload Data forźPerformanceźEvaluation“. In: *Proceedings of the 22Nd International Conference on Euro-Par 2016: Parallel Processing - Volume 9833*. New York, NY, USA: Springer-Verlag New York, Inc., 2016, pp. 3–21 (cit. on p. 167).

[Fel+15] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. „An updated performance comparison of virtual machines and Linux containers“. In: *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2015, pp. 171–172 (cit. on pp. 33, 44–46).

[FO96] John L. Furlani and Peter W. Osel. „Abstract Yourself With Modules“. In: *Proceedings of the 10th USENIX Conference on System Administration*. LISA ’96. Chicago, IL: USENIX Association, 1996, pp. 193–204 (cit. on p. 53).

[FR96] Dror G. Feitelson and Larry Rudolph. „Toward convergence in job schedulers for parallel supercomputers“. In: *Job Scheduling Strategies for Parallel Processing*. Ed. by Dror G. Feitelson and Larry Rudolph. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 1–26 (cit. on pp. 63, 68).

[Gaf+14a] N. Gaffney, C. Jordan, T. Minyard, and D. Stanzione. „Building Wrangler: A transformational data intensive resource for the open science community“. In: *2014 IEEE International Conference on Big Data (Big Data)*. 2014, pp. 20–22 (cit. on p. 23).

[Gaf+14b] Niall Gaffney, Christopher Jordan, Tommy Minyard, and Dan Stanzione. „Building wrangler: A transformational data intensive resource for the open science community“. In: *Big Data (Big Data), 2014 IEEE International Conference on*. IEEE, 2014, pp. 20–22 (cit. on p. 63).

[Gam+15] Todd Gamblin, Matthew LeGendre, Michael R. Collette, et al. „The Spack Package Manager: Bringing Order to HPC Software Chaos“. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '15. Austin, Texas: ACM, 2015, 40:1–40:12 (cit. on p. 54).

[Gao+18] Wanling Gao, Lei Wang, Jianfeng Zhan, et al. „Big Data Dwarfs: Towards Fully Understanding Big Data Analytics Workloads“. In: *CoRR* abs/1802.00699 (2018). arXiv: 1802.00699 (cit. on p. 13).

[Geo10] Yiannis Ioannis Georgiou. „Contributions for Resource and Job Management in High Performance Computing“. Theses. Université de Grenoble, Nov. 2010 (cit. on p. 24).

[GGL03] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. „The Google File System“. In: *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*. SOSP '03. Bolton Landing, NY, USA: ACM, 2003, pp. 29–43 (cit. on p. 8).

[GHM14] M. Geimer, K. Hoste, and R. McLay. „Modern Scientific Software Management Using EasyBuild and Lmod“. In: *2014 First International Workshop on HPC User Support Tools*. Nov. 2014, pp. 41–51 (cit. on p. 54).

[Git+16] Alex Gittens, Aditya Devarakonda, Evan Racah, et al. „Matrix Factorizations at Scale: a Comparison of Scientific Data Analytics in Spark and C+MPI Using Three Case Studies“. In: (May 2016) (cit. on p. 18).

[GKN18] Cristian Galleguillos, Zeynep Kiziltan, and Alessio Netti. „AccaSim: An HPC Simulator for Workload Management“. In: *High Performance Computing*. Ed. by Esteban Mocskos and Sergio Nesmachnow. Cham: Springer International Publishing, 2018, pp. 169–184 (cit. on p. 92).

[GNQ13] Maximiliano Geier, Lucas Nussbaum, and Martin Quinson. „On the Convergence of Experimental Methodologies for Distributed Systems: Where do we stand?“ In: *WATERS - 4th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*. Paris, France, July 2013 (cit. on p. 154).

[Gog+16] Ionel Gog, Malte Schwarzkopf, Adam Gleave, Robert NM Watson, and Steven Hand. „Firmament: fast, centralized cluster scheduling at scale“. In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 2016, pp. 99–115 (cit. on pp. 92, 105).

[Gom+17a] Jorge Gomes, Isabel Campos Plasencia, Emanuele Bagnaschi, et al. „Enabling rootless Linux Containers in multi-user environments: the udocker tool“. In: *CoRR* abs/1711.01758 (2017). arXiv: 1711.01758 (cit. on p. 33).

[Gom+17b] Jorge Gomes, Isabel Campos Plasencia, Emanuele Bagnaschi, et al. „Enabling rootless Linux Containers in multi-user environments: the udocker tool“. In: *CoRR* abs/1711.01758 (2017). arXiv: 1711.01758 (cit. on pp. 48, 49).

[GRC07] Yiannis Georgiou, Olivier Richard, and Nicolas Capit. „Evaluations of the Lightweight Grid CIGRI upon the Grid5000 Platform“. In: *Third International Conference on e-Science and Grid Computing, e-Science 2007, 10-13 December 2007, Bangalore, India*. 2007, pp. 279–286 (cit. on p. 69).

[GS09] Julius Gehr and Jörg Schneider. „Measuring Fragmentation of Two-Dimensional Resources Applied to Advance Reservation Grid Scheduling“. In: IEEE, 2009, pp. 276–283 (cit. on p. 87).

[Han+09] J. E. Hannay, C. MacLeod, J. Singer, et al. „How do scientists develop and use scientific software?“ In: *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*. May 2009, pp. 1–8 (cit. on p. 125).

[HB16] James Howison and Julia Bullard. „Software in the scientific literature: Problems with seeing, finding, and using software mentioned in the biology literature“. In: *Journal of the Association for Information Science and Technology* 67.9 (2016), pp. 2137–2155. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/asi.23538 (cit. on p. 125).

[HDB17] Melanie Herschel, Ralf Diestelkämper, and Houssem Ben Lahmar. „A Survey on Provenance: What for? What Form? What from?“ In: *The VLDB Journal* 26.6 (Dec. 2017), pp. 881–906 (cit. on p. 146).

[He+15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. „Deep Residual Learning for Image Recognition“. In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385 (cit. on p. 14).

[Hei+17a] Franz C. Heinrich, Alexandra Carpen-Amarie, Augustin Degomme, et al. „Predicting the Performance and the Power Consumption of MPI Applications With SimGrid“. working paper or preprint. Jan. 2017 (cit. on p. 121).

[Hei+17b] Franz C. Heinrich, Tom Cornebize, Augustin Degomme, et al. „Predicting the Energy Consumption of MPI Applications at Scale Using a Single Node“. In: *Cluster 2017*. IEEE. Hawaii, United States, Sept. 2017 (cit. on p. 121).

[Her+16] Stephen Herbein, Dong H. Ahn, Don Lipari, et al. „Scalable I/O-Aware Job Scheduling for Burst Buffer Enabled HPC Clusters“. In: *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*. HPDC '16. Kyoto, Japan: ACM, 2016, pp. 69–80 (cit. on pp. 92, 106).

[Hie18] PHAM Tuan Hiep. „Study the impact of Big Data File System on High Performance Computing Applications“. https://gitlab.inria.fr/bebida/internship/IO-impact/blob/master/Hiep_PHAM_report_DFS_impact_on_HPC_app_REPORT.pdf. MA thesis. Univ. Grenbole Alpes, Sept. 2018 (cit. on pp. 31, 76).

[Hin+11] Benjamin Hindman, Andy Konwinski, Matei Zaharia, et al. „Mesos: A Platform for Fine-grained Resource Sharing in the Data Center“. In: *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*. NSDI'11. Boston, MA: USENIX Association, 2011, pp. 295–308 (cit. on pp. 27, 68, 118).

[Imb+13] Matthieu Imbert, Laurent Pouilloux, Jonathan Rouzaud-Cornabas, Adrien Lèbre, and Takahiro Hirofuchi. „Using the EXECO toolbox to perform automatic and reproducible cloud experiments“. In: *1st International Workshop on UsiNg and building ClOud Testbeds (UNICO), collocated with IEEE CloudCom 2013*. Bristol, United Kingdom: IEEE, Dec. 2013 (cit. on pp. 77, 149).

[Isl+15] N. S. Islam, X. Lu, M. Wasi-ur-Rahman, D. Shankar, and D. K. Panda. „Triple-H: A Hybrid Approach to Accelerate HDFS on HPC Clusters with Heterogeneous Storage Architecture“. In: *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. May 2015, pp. 101–110 (cit. on p. 30).

[JC15] Douglas M Jacobsen and Richard Shane Canon. „Contain this, unleashing docker for hpc“. In: *Proceedings of the Cray User Group* (2015) (cit. on pp. 33, 49).

[Jim+17] Ivo Jimenez, Michael Sevilla, Noah Watkins, et al. „The popper convention: Making reproducible systems evaluation practical“. In: *Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017 IEEE International*. IEEE. 2017, pp. 1561–1570 (cit. on pp. 132, 142, 147).

[Kas+15] Sidharth N Kashyap, Ade J Fewings, Jay Davies, et al. „Big Data at HPC Wales“. In: *arXiv preprint arXiv:1506.08907* (2015) (cit. on pp. 71, 74).

[Kep+15] Jeremy Kepner, William Arcand, David Bestor, et al. „Lustre, Hadoop, Accumulo“. In: *CoRR* abs/1507.02357 (2015). arXiv: 1507.02357 (cit. on p. 28).

[KGP89] R. H. Katz, G. A. Gibson, and D. A. Patterson. „Disk system architectures for high performance computing“. In: *Proceedings of the IEEE* 77.12 (1989), pp. 1842–1858 (cit. on p. 7).

[Knu84] D. E. Knuth. „Literate Programming“. In: *The Computer Journal* 27.2 (Jan. 1984), pp. 97–111. eprint: http://oup.prod.sis.lan/comjnl/article-pdf/27/2/97/981657/270097.pdf (cit. on p. 138).

[KSB17a] Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. „Singularity: Scientific containers for mobility of compute“. In: *PLOS ONE* 12.5 (May 2017), pp. 1–20 (cit. on p. 47).

[KSB17b] Gregory M Kurtzer, Vanessa Sochat, and Michael W Bauer. „Singularity: Scientific containers for mobility of compute“. In: *PloS one* 12.5 (2017), e0177459 (cit. on p. 33).

[KTB11] Sriram Krishnan, Mahidhar Tatineni, and Chaitanya Baru. „myHadoop-Hadoop-on-Demand on traditional HPC resources“. In: *San Diego Supercomputer Center Technical Report TR-2011-2, University of California, San Diego* (2011) (cit. on pp. 9, 70).

[KTP16] Dalibor Klusaček, Šimon Toth, and Gabriela Podolnıkova. „Complex Job Scheduling Simulations with Alea 4“. In: *Proceedings of the 9th EAI International Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 2016, pp. 124–129 (cit. on pp. 92, 154).

[Law+01] Steve Lawrence, David M Pennock, Gary William Flake, et al. „Persistence of web references in scientific research“. In: *Computer* 2 (2001), pp. 26–31 (cit. on p. 134).

[LC16] Henry Z. Lo and Joseph Paul Cohen. „Academic Torrents: Scalable Data Distribution“. In: *CoRR* abs/1603.04395 (2016). arXiv: 1603.04395 (cit. on p. 147).

[Liu+12] Ning Liu, Jason Cope, Philip Carns, et al. „On the Role of Burst Buffers in Leadership-class Storage Systems“. In: Apr. 2012 (cit. on pp. 93, 96).

[Liu+15] N. Liu, X. Yang, X. Sun, J. Jenkins, and R. Ross. „YARNsim: Simulating Hadoop YARN“. In: *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. 2015, pp. 637–646 (cit. on p. 93).

[Liv+97] Miron Livny, Jim Basney, Rajesh Raman, and Todd Tannenbaum. „Mechanisms for high throughput computing“. In: *SPEEDUP journal* 11.1 (1997), pp. 36–40 (cit. on p. 8).

[Luc+16] Andre Luckow, Ioannis Paraskevakos, George Chantzialexiou, and Shantenu Jha. „Hadoop on HPC: Integrating Hadoop and Pilot-based Dynamic Resource Management“. In: *arXiv preprint arXiv:1602.00345* (2016) (cit. on pp. 10, 30, 73, 74).

[Ma09] K. Ma. „In Situ Visualization at Extreme Scale: Challenges and Opportunities“. In: *IEEE Computer Graphics and Applications* 29.6 (Nov. 2009), pp. 14–19 (cit. on p. 16).

[Mar+16] Ovidiu Marcu, Alexandru Costan, Gabriel Antoniu, and María Pérez. „Spark Versus Flink: Understanding Performance in Big Data Analytics Frameworks“. In: Sept. 2016, pp. 433–442 (cit. on p. 21).

[McL+11] Robert McLay, Karl W. Schulz, William L. Barth, and Tommy Minyard. „Best Practices for the Deployment and Management of Production HPC Clusters“. In: *State of the Practice Reports*. SC ’11. Seattle, Washington: ACM, 2011, 9:1–9:11 (cit. on p. 53).

[MCY15] Joe Mambretti, Jim Chen, and Fei Yeh. „Next Generation Clouds, the Chameleon Cloud Testbed, and Software Defined Networking (SDN)“. In: *Proceedings of the 2015 International Conference on Cloud Computing Research and Innovation (ICCCRI)*. ICCCRI ’15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 73–79 (cit. on pp. 52, 168).

[MER+17] Michael MERCIER, David Glesser, Yiannis Georgiou, and Olivier Richard. „Big Data and HPC collocation: Using HPC idle resources for Big Data Analytics“. In: *IEEE BigData 2017*. Boston, United States, Dec. 2017 (cit. on p. 62).

[MFR18] Michael Mercier, Adrien Faure, and Olivier Richard. „Considering the Development Workflow to Achieve Reproducibility with Variation“. In: *SC 2018 - Workshop: ResCuE-HPC*. Dallas, United States, Nov. 2018, pp. 1–5 (cit. on p. 127).

[Min+13] Zijian Ming, Chunjie Luo, Wanling Gao, et al. „BDGS: A scalable big data generator suite in big data benchmarking“. In: *Workshop on Big Data Benchmarks*. Springer, 2013, pp. 138–154 (cit. on p. 79).

[Mol+09] Esteban Molina-Estolano, Maya Gokhale, Carlos Maltzahn, et al. „Mixing Hadoop and HPC Workloads on Parallel Filesystems“. In: *Proceedings of the 4th Annual Workshop on Petascale Data Storage*. PDSW ’09. Portland, Oregon: ACM, 2009, pp. 1–5 (cit. on p. 29).

[Moo+13] W. C. Moody, L. B. Ngo, E. Duffy, and A. Apon. „JUMMP: Job Uninterrupted Maneuverable MapReduce Platform“. In: *2013 IEEE International Conference on Cluster Computing (CLUSTER)*. 2013, pp. 1–8 (cit. on p. 10).

[NAS18] NASA Ames Research Center NASA Advanced Supercomputing Division. *Evaluating the Suitability of Commercial Clouds for NASA’s High Performance Computing Applications: A Trade Study*. May 2018 (cit. on p. 33).

[Nas90] Stephen G. Nash, ed. *A History of Scientific Computing*. New York, NY, USA: ACM, 1990 (cit. on pp. 6, 8).

[NFD12] Marcelo Veiga Neves, Tiago Ferreto, and César De Rose. „Scheduling mapreduce jobs in hpc clusters“. In: *Euro-Par 2012 Parallel Processing*. Springer, 2012, pp. 179–190 (cit. on p. 71).

[NSJ04] Bill Nitzberg, Jennifer M Schopf, and James Patton Jones. „PBS Pro: Grid computing and scheduling attributes“. In: *Grid resource management*. 2004 (cit. on p. 68).

[Nüs+18] Daniel Nüst, Carlos Granell, Barbara Hofer, et al. „Reproducible research and GIScience: an evaluation using AGILE conference papers“. In: *PeerJ* 6 (July 2018), e5072 (cit. on p. 136).

[Ous+15] Kay Ousterhout, Ryan Rasti, Sylvia Ratnasamy, et al. „Making Sense of Performance in Data Analytics Frameworks.“ In: *NSDI*. Vol. 15. 2015, pp. 293–307 (cit. on p. 19).

[PD11] Steven J. Plimpton and Karen D. Devine. „MapReduce in MPI for Large-scale Graph Algorithms“. In: *Parallel Comput.* 37.9 (Sept. 2011), pp. 610–632 (cit. on p. 69).

[Pen+17] Bo Peng, Bingjing Zhang, Langshi Chen, et al. „HarpLDA+: Optimizing latent dirichlet allocation for parallel efficiency“. In: *Big Data (Big Data), 2017 IEEE International Conference on*. IEEE. 2017, pp. 243–252 (cit. on p. 20).

[PNM09] Jose Antonio Pascual, Javier Navaridas, and Jose Miguel-Alonso. „Effects of Topology-Aware Allocation Policies on Scheduling Performance“. In: *Job Scheduling Strategies for Parallel Processing*. Ed. by Eitan Frachtenberg and Uwe Schwiegelshohn. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 138–156 (cit. on p. 92).

[Pop59] Karl Raimund Popper. *The Logic of Scientific Discovery*. Routledge, 1959 (cit. on p. 126).

[PR17a] Reid Priedhorsky and Tim Randles. „Charliecloud: Unprivileged Containers for User-defined Software Stacks in HPC“. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC ’17. Denver, Colorado: ACM, 2017, 36:1–36:10 (cit. on p. 48).

[PR17b] Reid Priedhorsky and Tim Randles. „Charliecloud: Unprivileged containers for user-defined software stacks in hpc“. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM. 2017, p. 36 (cit. on p. 33).

[Rat85] Justin Rattner. „Concurrent processing: A new direction in scientific computing“. In: *Proc. AFIPS Conf.* Vol. 54. 1985, pp. 157–166 (cit. on p. 7).

[RDS16] Ramya Raghavendra, Pranita Dewan, and Mudhakar Srivatsa. „Unifying HDFS and GPFS: Enabling Analytics on Software-Defined Storage“. In: *Proceedings of the 17th International Middleware Conference*. Middleware ’16. Trento, Italy: ACM, 2016, 3:1–3:13 (cit. on p. 30).

[Ren+13] Kai Ren, YongChul Kwon, Magdalena Balazinska, and Bill Howe. „Hadoop’s Adolescence: An Analysis of Hadoop Usage in Scientific Workloads“. In: *Proc. VLDB Endow.* 6.10 (Aug. 2013), pp. 853–864 (cit. on p. 13).

[RFB11] Ioan Raicu, Ian T. Foster, and Pete Beckman. „Making a Case for Distributed File Systems at Exascale“. In: *Proceedings of the Third International Workshop on Large-scale System and Application Performance*. LSAP ’11. San Jose, California, USA: ACM, 2011, pp. 11–18 (cit. on p. 31).

[Ric+15] Robert Ricci, Gary Wong, Leigh Stoller, et al. „Apt: A platform for repeatable research in computer science“. In: *ACM SIGOPS Operating Systems Review* 49.1 (2015), pp. 100–107 (cit. on p. 132).

[RJN15] Cristian Ruiz, Emmanuel Jeanvoine, and Lucas Nussbaum. „Performance evaluation of containers for HPC“. In: *VHPC - 10th Workshop on Virtualization in High-Performance Cloud Computing*. VHPC - 10th Workshop on Virtualization in High-Performance Cloud Computing. Vienna, Austria, Aug. 2015, p. 12 (cit. on p. 33).

[ROA15a] Jorge L. Reyes-Ortiz, Luca Oneto, and Davide Anguita. „Big Data Analytics in the Cloud: Spark on Hadoop vs MPI/OpenMP on Beowulf“. In: *Procedia Computer Science* 53 (2015). INNS Conference on Big Data 2015 Program San Francisco, CA, USA 8-10 August 2015, pp. 121–130 (cit. on p. 19).

[ROA15b] Jorge L. Reyes-Ortiz, Luca Oneto, and Davide Anguita. „Big Data Analytics in the Cloud: Spark on Hadoop vs MPI/OpenMP on Beowulf“. In: *Procedia Computer Science*. INNS Conference on Big Data 2015 Program San Francisco, CA, USA 8-10 August 2015 53 (Jan. 2015), pp. 121–130 (cit. on p. 69).

[RSR15] Johanna Renker, Stephan Schlagkamp, and Gerhard Rinkenauer. „Questionnaire for User Habits of Compute Clusters (QUHCC)“. In: Aug. 2015, pp. 697–702 (cit. on p. 167).

[Rui+15] Cristian Ruiz, Salem Harrache, Michael Mercier, and Olivier Richard. „Reconstructable Software Appliances with Kameleon“. In: *Operating Systems Review* 49.1 (2015), pp. 80–89 (cit. on pp. 52, 77, 135, 149, 152).

[S A16] Mohammed S. Al-kahtani. „Big Data Networking : Requirements, Architecture and Issues“. In: *International Journal of Wireless & Mobile Networks* 8 (Dec. 2016), pp. 35–49 (cit. on p. 32).

[Sch+13] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. „Omega: flexible, scalable schedulers for large compute clusters“. In: *SIGOPS European Conference on Computer Systems (EuroSys)*. Prague, Czech Republic, 2013, pp. 351–364 (cit. on p. 68).

[Ser+18] Marc Sergent, Mario Dagrada, Patrick Carribault, et al. „Efficient Communication/Computation Overlap with MPI+OpenMP Runtimes Collaboration“. In: *Euro-Par 2018: Parallel Processing - 24th International Conference on Parallel and Distributed Computing, Turin, Italy, August 27-31, 2018, Proceedings*. 2018, pp. 560–572 (cit. on p. 166).

[SLD15] Luka Stanisic, Arnaud Legrand, and Vincent Danjean. „An Effective Git And Org-Mode Based Workflow For Reproducible Research“. In: *SIGOPS Oper. Syst. Rev.* 49.1 (Jan. 2015), pp. 61–70 (cit. on p. 142).

[Sul+08] Anthony Sulistio, Uros Cibej, Srikumar Venugopal, Borut Robic, and Rajkumar Buyya. „A toolkit for modelling and simulating data Grids: an extension to GridSim“. In: *Concurrency and Computation: Practice and Experience* 20.13 (2008), pp. 1591–1609 (cit. on p. 154).

[Sur+10] Sayantan Sur, Hao Wang, Jian Huang, Xiangyong Ouyang, and Dhabaleswar K Panda. „Can high-performance interconnects benefit hadoop distributed file system“. In: *Workshop on Micro Architectural Support for Virtualization, Data Center Computing, and Clouds (MASVDC). Held in Conjunction with MICRO*. Citeseer. 2010 (cit. on p. 10).

[Tan+11] W. Tantisiriroj, S. W. Son, S. Patil, et al. „On the duality of data-intensive file system design: Reconciling HDFS and PVFS“. In: *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. Nov. 2011, pp. 1–12 (cit. on p. 30).

[TAS17] Ehsan Totoni, Todd A Anderson, and Tatiana Shpeisman. „HPAT: high performance analytics with scripting ease-of-use“. In: *Proceedings of the International Conference on Supercomputing*. ACM. 2017, p. 9 (cit. on p. 20).

[Tro+07] P. Troger, H. Rajic, A. Haas, and P. Domagalski. „Standardization of an API for Distributed Resource Management Systems“. In: *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07)*. May 2007, pp. 619–626 (cit. on p. 166).

[Use+10] Andrew Uselton, Mark Howison, Nicholas J. Wright, et al. „Parallel I/O performance: From events to ensembles“. In: *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–11 (cit. on pp. 29, 86, 90).

[Vav+13a] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, et al. „Apache Hadoop YARN: Yet Another Resource Negotiator“. In: *Proceedings of the 4th Annual Symposium on Cloud Computing*. SOCC '13. Santa Clara, California: ACM, 2013, 5:1–5:16 (cit. on pp. 68, 118).

[Vav+13b] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, et al. „Apache hadoop yarn: Yet another resource negotiator“. In: *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM. 2013, p. 5 (cit. on pp. 9, 68).

[Vel+13] Pedro Velho, Lucas Schnorr, Henri Casanova, and Arnaud Legrand. „On the Validity of Flow-level TCP Network Models for Grid and Cloud Simulations“. In: *ACM Transactions on Modeling and Computer Simulation* 23.4 (Oct. 2013) (cit. on p. 93).

[Vil+08] Murali Vilayannur, Samuel Lang, Robert Ross, Ruth Klundt, Lee Ward, et al. „Extending the POSIX I/O interface: A parallel file system perspective". In: *Argonne National Laboratory, Tech. Rep. ANL/MCS-TM-302* 10 (2008) (cit. on p. 29).

[Wan+14] Lei Wang, Jianfeng Zhan, Chunjie Luo, et al. „Bigdatabench: A big data benchmark suite from internet services". In: *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*. IEEE, 2014, pp. 488–499 (cit. on p. 79).

[Wan+18] Xin Wang, Misbah Mubarak, Xu Yang, Robert B Ross, and Zhiling Lan. „Trade-Off Study of Localizing Communication and Balancing Network Traffic on a Dragonfly System". In: *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2018, pp. 1113–1122 (cit. on p. 154).

[Was+15] M. Wasi-ur-Rahman, Xiaoyi Lu, N.S. Islam, R. Rajachandrasekar, and D.K. Panda. „High-Performance Design of YARN MapReduce on Modern HPC Clusters with Lustre and RDMA". In: *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*. May 2015, pp. 291–300 (cit. on p. 69).

[Whi+02] Brian White, Jay Lepreau, Leigh Stoller, et al. „An Integrated Experimental Environment for Distributed Systems and Networks". In: "*Proc. of the Fifth Symposium on Operating Systems Design and Implementation*". USENIX Association. Boston, MA, Dec. 2002, pp. 255–270 (cit. on pp. 52, 168).

[Wur+18] Ricardo Wurmus, Bora Uyar, Brendan Osberg, et al. „Reproducible genomics analysis pipelines with GNU Guix". In: *bioRxiv* (2018). eprint: https://www.biorxiv.org/content/early/2018/04/21/298653.full.pdf (cit. on p. 133).

[Xu+17] Luna Xu, Seung-Hwan Lim, Min Li, Ali Raza Butt, and Ramakrishnan Kannan. „Scaling up data-parallel analytics platforms: Linear algebraic operation cases." In: *BigData*. 2017, pp. 273–282 (cit. on p. 19).

[Xua+15] Pengfei Xuan, Jeffrey Denton, Pradip K. Srimani, Rong Ge, and Feng Luo. „Big Data Analytics on Traditional HPC Infrastructure Using Two-level Storage". In: *Proceedings of the 2015 International Workshop on Data-Intensive Scalable Computing Systems*. DISCS '15. New York, NY, USA: ACM, 2015, 4:1–4:8 (cit. on pp. 29, 30, 63).

[YI18] Orcun Yildiz and Shadi Ibrahim. „On the Performance of Spark on HPC Systems: Towards a Complete Picture". In: *Supercomputing Frontiers*. Ed. by Rio Yokota and Weigang Wu. Cham: Springer International Publishing, 2018, pp. 70–89 (cit. on pp. 29, 117).

[YJG03] Andy B. Yoo, Morris A. Jette, and Mark Grondona. „SLURM: Simple Linux Utility for Resource Management". In: *JSSPP*. 2003 (cit. on p. 68).

[Zah+12] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, et al. „Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing“. In: *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 2–2 (cit. on pp. 17, 68, 81).

[Zan+18] Francieli Zanon Boito, Eduardo Camilo Inacio, Jean Luca Bez, et al. „A Checkpoint of Research on Parallel I/O for High Performance Computing“. In: *ACM Computing Surveys* 51.2 (Mar. 2018), 23:1–23:35 (cit. on p. 29).

[ZRQ15] Bingjing Zhang, Yang Ruan, and Judy Qiu. „Harp: Collective communication on hadoop“. In: *Cloud Engineering (IC2E), 2015 IEEE International Conference on*. IEEE. 2015, pp. 228–233 (cit. on pp. 10, 19).

## Webpages

[Ali19] Alibaba. *Derrick - documentation*. 2019. URL: `https://alibaba.github.io/derrick/` (visited on Feb. 5, 2019) (cit. on p. 133).

[Apa15] Apache. *Software projects built on Mesos*. Sept. 2015. URL: `http://mesos.apache.org/documentation/latest/mesos-frameworks/` (visited on Sept. 18, 2015) (cit. on pp. 72, 74).

[CER18] CERN. *Welcome to the Worldwide LHC Computing Grid*. 2018. URL: `http://wlcg.web.cern.ch/` (cit. on pp. 1, 14).

[CER19] CERN. *CASTOR at CERN statistics*. 2019. URL: `http://castorwww.web.cern.ch/castorwww/namespace_statistics.png` (visited on Feb. 5, 2019) (cit. on p. 9).

[Chu13] Al Chu. *chu11/magpie*. Oct. 2013. URL: `https://github.com/chu11/magpie` (visited on Sept. 16, 2015) (cit. on pp. 70, 74).

[com19a] NixOS community. *Disnix website*. 2019. URL: `https://nixos.org/disnix/` (visited on Feb. 6, 2019) (cit. on p. 168).

[com19b] NixOS community. *NixOps website*. 2019. URL: `https://nixos.org/nixops/` (visited on Feb. 6, 2019) (cit. on p. 168).

[com19c] Reproducible Builds community. *Reproducible Build*. 2019. URL: `https://reproducible-builds.org/` (visited on Feb. 6, 2019) (cit. on p. 167).

[Cou18] Ludovic Courtès. *GuixHPC web site*. 2018. URL: `https://guix-hpc.bordeaux.inria.fr/` (visited on May 22, 2018) (cit. on p. 58).

[CW18] Ludovic Courtès and Ricardo Wurmus. *Reproducible software deployment for high-performance computing*. 2018. URL: `http://hpc.guixsd.org/about/` (visited on June 14, 2018) (cit. on p. 58).

[CWL18] CWL. *Common Workflow Language*. 2018. URL: https://www.commonwl.org (visited on Sept. 3, 2018) (cit. on p. 145).

[Fac12] Facebook. *Corona*. 2012. URL: https://www.facebook.com/notes/facebook-engineering/under-the-hood-scheduling-mapreduce-jobs-more-efficiently-with-corona/10151142560538920/ (visited on Feb. 6, 2019) (cit. on p. 68).

[Fei17] Dror Feitelson. *Parallel Workloads Archive*. 2017. URL: %7Bhttp://www.cs.huji.ac.il/labs/parallel/workload/%7D (visited on Feb. 14, 2017) (cit. on pp. 11, 157).

[Fei18] Feitelson. *The Standard Workload Format version 2.2*. 2018. URL: http://www.cs.huji.ac.il/labs/parallel/workload/swf.html (visited on Sept. 4, 2018) (cit. on p. 157).

[Fig19] Figshare. *Figshare website*. 2019. URL: https://figshare.com/ (visited on Feb. 6, 2019) (cit. on p. 146).

[Fon17] Infiniband Trade Fondation. *InfiniBand Accelerates the World's Fastest Supercomputer, Two of the Top Five Supercomputers, and 77 Percent of New HPC Systems on the TOP500 List*. 2017. URL: https://www.infinibandta.org/infiniband-accelerates-the-worlds-fastest-supercomputer-two-of-the-top-five-supercomputers-and-77-percent-of-new-hpc-systems-on-the-top500-list/ (visited on Feb. 6, 2019) (cit. on p. 22).

[Fon18] Apache Fondation. *Spark current commiters*. 2018. URL: https://spark.apache.org/committers.html (cit. on p. 17).

[Fon19a] Apache Software Fondation. *Apache Flink*. 2019. URL: https://flink.apache.org/ (visited on Feb. 6, 2019) (cit. on p. 63).

[Fon19b] Apache Software Fondation. *Apache Hadoop*. 2019. URL: https://hadoop.apache.org/ (visited on Feb. 6, 2019) (cit. on pp. 24, 63).

[Fon19c] Apache Software Fondation. *Apache Spark*. 2019. URL: https://spark.apache.org/ (visited on Feb. 6, 2019) (cit. on pp. 63, 101).

[git18a] git-annex. *Git Large File Storage*. 2018. URL: https://git-lfs.github.com/ (visited on Feb. 6, 2019) (cit. on p. 134).

[git18b] git-annex. *git-annex*. 2018. URL: https://git-annex.branchable.com/ (visited on Feb. 6, 2019) (cit. on p. 134).

[Gri18] Grid'5000. *Grid5000 wiki - Experiment scripting tutorial*. 2018. URL: https://www.grid5000.fr/mediawiki/index.php/Experiment_scripting_tutorial (visited on May 23, 2018) (cit. on pp. 52, 146).

[Gui19] GNU Guile. *Guile is a programming language*. 2019. URL: https://www.gnu.org/software/guile/ (visited on Feb. 5, 2019) (cit. on p. 57).

[Has19] HashiCorp. *Terraform website*. 2019. URL: https://www.terraform.io/ (visited on Feb. 6, 2019) (cit. on p. 168).

[Hat19a] Red Hat. *ansible website*. 2019. URL: https://www.ansible.com/ (visited on Feb. 6, 2019) (cit. on p. 168).

[Hat19b] Red Hat. *SaltStack website*. 2019. URL: https://www.saltstack.com/ (visited on Feb. 6, 2019) (cit. on p. 168).

[Hin18] Jonathan Hines. *ORNL researchers leverage GPU Tensor Cores to deliver unprecedented performance*. 2018. URL: https://www.olcf.ornl.gov/2018/06/08/genomics-code-exceeds-exaops-on-summit-supercomputer/ (visited on June 8, 2018) (cit. on p. 31).

[ICL18] ICL. *High Performance Conjugate Gradients*. 2018. URL: http://www.hpcg-benchmark.org/index.html (cit. on p. 12).

[Inc19] PeerJ Inc. *PeerJ website*. 2019. URL: https://peerj.com (visited on Feb. 6, 2019) (cit. on p. 147).

[Int17] Intel. *intel-hpdd/scheduling-connector-for-hadoop: HPC Adapter for Mapreduce/Yarn(HAM)*. July 19, 2017. URL: https://github.com/whamcloud/scheduling-connector-for-hadoop (cit. on pp. 71, 74, 118).

[Int18a] Intel. *Intel® Data Analytics Acceleration Library*. 2018. URL: https://software.intel.com/en-us/blogs/daal (cit. on p. 20).

[Int18b] Intel. *IntelLabs HPAT*. 2018. URL: https://github.com/IntelLabs/hpat (cit. on p. 20).

[Jup19] Project Jupyter. *Jupyter Project*. 2019. URL: https://www.gnu.https://jupyter.org/ (visited on Feb. 6, 2019) (cit. on pp. 145, 151).

[Lab19] Icalia Labs. *Whales - repostory*. 2019. URL: https://github.com/Gueils/whales (visited on Feb. 5, 2019) (cit. on p. 133).

[Leg18] Arnaud Legrand. *Simgrid Usages*. 2018. URL: http://simgrid.gforge.inria.fr/Usages.html (visited on Aug. 12, 2018) (cit. on p. 148).

[NAS16] NASA. *NAS Parallel Benchmarks*. Feb. 2016. URL: https://www.nas.nasa.gov/publications/npb.html (visited on Feb. 18, 2016) (cit. on pp. 12, 79).

[Ner18] Nersc. *NERSC-8 / Trinity Benchmarks*. 2018. URL: https://www.nersc.gov/users/computational-systems/cori/nersc-8-procurement/trinity-nersc-8-rfp/nersc-8-trinity-benchmarks/ (cit. on p. 12).

[OAK18] Livermore OAK Ridge Argonne. *Coral-2 Benchmarks*. 2018. URL: https://asc.llnl.gov/coral-2-benchmarks/index.php (cit. on p. 12).

[Opea] OpenStack. *OpenStack diskimage-builder web site*. URL: https://docs.openstack.org/diskimage-builder/latest/ (visited on May 23, 2018) (cit. on p. 52).

[Opeb] OpenStack. *OpenStack web site*. URL: https://www.openstack.org/ (visited on May 23, 2018) (cit. on p. 52).

[Ope19] RedHat - OpenShift. *S2I - documentation*. 2019. URL: https://github.com/openshift/source-to-image (visited on Feb. 5, 2019) (cit. on p. 133).

[org19] orgmode.org. *Org mode for Emacs*. 2019. URL: https://orgmode.org/ (visited on Feb. 6, 2019) (cit. on pp. 145, 151).

[Pri15] Timothy Prickett Morgan. *Teaching Grid Engine To Speak Mesos*. Apr. 2015. URL: http://www.theplatform.net/2015/04/21/teaching-grid-engine-to-speak-mesos/ (visited on Sept. 16, 2015) (cit. on pp. 72, 74).

[Pro18] Exascale Project. *Exascale Proxy Applications*. 2018. URL: https://proxyapps.exascaleproject.org/ (cit. on pp. 12, 122).

[Pro19a] GNU Project. *GNU Emacs*. 2019. URL: https://www.gnu.org/software/emacs/ (visited on Feb. 6, 2019) (cit. on p. 145).

[Pro19b] Jupyter Project. *jupyter-repo2docker - documentation*. 2019. URL: https://repo2docker.readthedocs.io (visited on Feb. 5, 2019) (cit. on p. 133).

[Pup19] Puppet. *Puppet website*. 2019. URL: https://puppet.com/ (visited on Feb. 6, 2019) (cit. on p. 168).

[Ste03] Thomas Sterling. *Beowulf Breakthroughs: The Path to Commodity Supercomputing*. 2003. URL: http://www.linux-mag.com/id/1378/ (visited on Feb. 5, 2019) (cit. on p. 7).

[Ste19] Stencila. *Dockter*. 2019. URL: https://stenci.la/learn/integrations/dockter.html (visited on Feb. 5, 2019) (cit. on p. 133).

[TAC18] TACC. *TACC Wrangler User Guide*. 2018. URL: https://portal.tacc.utexas.edu/user-guides/wrangler (cit. on p. 23).

[Top18] Top500. *Top500 statistics*. 2018. URL: %7Bhttps://top500.org/statistics/list/%7D (visited on May 4, 2018) (cit. on pp. 7, 43, 46).

[top19] top500.org. *Performance Development*. 2019. URL: https://www.top500.org/statistics/perfdevel/ (visited on Jan. 21, 2019) (cit. on p. 3).

[Uni18] Indiana University. *What is Harp-DAAL*. 2018. URL: https://dsc-spidal.github.io/harp/docs/harpdaal/harpdaal/ (cit. on pp. 10, 20).

[Uni19] Univia. *URB*. 2019. URL: https://github.com/UnivaCorporation/urb-core (visited on Feb. 6, 2019) (cit. on p. 166).

[Wik18] Wikipedia. *CERN wikipedia page*. 2018. URL: https://en.wikipedia.org/wiki/CERN (cit. on p. 15).

[Wik19a] Lustre Wiki. *Lustre Object Storage Service (OSS)*. 2019. URL: http://wiki.lustre.org/Lustre_Object_Storage_Service_(OSS) (visited on Feb. 5, 2019) (cit. on p. 28).

[Wik19b] Wikipedia. *DevOps - wikipedia article*. 2019. URL: https://en.wikipedia.org/wiki/DevOps (visited on Feb. 6, 2019) (cit. on p. 142).

[Wik19c] Wikipedia. *Internet traffic - Global Internet traffic*. 2019. URL: https://en.wikipedia.org/wiki/Internet_traffic#Global_Internet_traffic (visited on Jan. 21, 2019) (cit. on p. 2).

[Zen19] Zenodo. *Zenodo website*. 2019. URL: https://zenodo.org/ (visited on Feb. 6, 2019) (cit. on p. 146).

# List of Figures

# List of Tables