HAL
open science

Workload Traces Analysis and Replay in Large Scale Distributed Systems

Joseph Emeras

To cite this version:

Joseph Emeras. Workload Traces Analysis and Replay in Large Scale Distributed Systems. Networking and Internet Architecture [cs.NI]. Université de Grenoble, 2013. English. NNT : 2013GRENM081 . tel-00940055

HAL Id: tel-00940055

https://theses.hal.science/tel-00940055v1

Submitted on 31 Jan 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministériel : 7 août 2006

Présentée par

Joseph Emeras

Thèse dirigée par **Yves Denneulin**
co-dirigée par **Olivier Richard**
co-encadrée par **Philippe Deniel**

préparée au sein du **Laboratoire d'Informatique de Grenoble**
et de l'**École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Workload Traces Analysis and Replay in Large Scale Distributed Systems

Thèse soutenue publiquement le **01 Octobre 2013**,
devant le jury composé de :

M., Denis Trystram
Professeur à Grenoble INP, Président
M., Dror Feitelson
Professeur à The Hebrew University, Jerusalem, Rapporteur
M., Christian Perez
Directeur de Recherche à INRIA, Rapporteur
Mme, Christine Morin
Directeur de Recherche à INRIA, Examinateur
M., Philippe Deniel
Ingénieur au CEA-DAM, Examinateur
M., Yves Denneulin
Professeur à Grenoble INP, Directeur de thèse
M., Olivier Richard
Maitre de conférences à l'UJF, Co-Directeur de thèse

Workload Traces Analysis and Replay in Large Scale Distributed Systems

A Dissertation
submitted to the department of
Computer Sciences
and the committee on graduate studies of
Grenoble University
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Joseph Emeras

October 2013

Abstract

High Performance Computing is preparing the era of the transition from Petascale to Exascale. Distributed computing systems are already facing new scalability problems due to the increasing number of computing resources to manage. It is now necessary to study in depth these systems and comprehend their behaviors, strengths and weaknesses to better build the next generation. The complexity of managing users applications on the resources conducted to the analysis of the workload the platform has to support, this to provide them an efficient service.

The need for workload comprehension has led to the collection of traces from production systems and to the proposal of a standard workload format. These contributions enabled the study of numerous of these traces. This also led to the construction of several models, based on the statistical analysis of the different workloads from the collection. Until recently, existing workload traces did not enabled researchers to study the consumption of resources by the jobs in a temporal way. This is now changing with the need for characterization of jobs consumption patterns.
In the first part of this thesis we propose a study of existing workload traces. Then we contribute with an observation of cluster workloads with the consideration of the jobs resource consumptions over time. This highlights specific and unattended patterns in the usage of resources from users. Finally, we propose an extension of the former standard workload format that enables to add such temporal consumptions without loosing the benefit of the existing works.

Experimental approaches based on workload models have also served the goal of distributed systems evaluation. Existing models describe the average behavior of observed systems. However, although the study of average behaviors is essential for the understanding of distributed systems, the study of critical cases and particular scenarios is also necessary. This study would give a more complete view and understanding of the performance of resource and job management.
In the second part of this thesis we propose an experimental method for performance evaluation of distributed systems based on the replay of production workload trace extracts. These extracts, replaced in their original context, enable to experiment the change of configuration of the system in an online workload and observe the different configurations results. Our technical contribution in this experimental approach is twofold. We propose a first tool to construct the environment in which the experimentation will take place, then we propose a second set of tools that automatize the experiment setup and that replay the trace extract within its original context.

Finally, these contributions conducted together, enable to gain a better knowledge of HPC platforms. As future works, the approach proposed in this thesis will serve as a basis to further study larger infrastructures.

Acknowledgements

First I would like to thank the institutions that funded this PhD thesis: CEA and CNRS, and also INRIA for hosting me in the MESCAL project during this period. I would like to thank my advisors at INRIA and CEA: Yves Denneulin and Philippe Deniel, and in a more personal point of view Olivier Richard my co-advisor that supported me all these years and for the good times, working or having fun. I also want to thank Yiannis Georgiou for having inspiring me the idea of doing this thesis.

Finally, I thank all the people that participated and were involved directly or indirectly in this project, whether they are colleagues, family or friends.

Willingness opens the doors to knowledge, direction, and achievement.

Chinese Fortune Cookie.

Contents

List of Figures **iii**

List of Tables **v**

1 Introduction **1**
1.1 Motivation 2
1.2 Distributed Systems Performance Evaluation 3
1.3 Workload Traces Analysis 7
1.4 Experimental Evaluation 8
1.5 Evaluation Process 8
1.6 Contributions 9
1.7 Thesis Overview 11

I Workload Traces Analysis **13**

1 State of the Art **15**
1.1 Cluster Workload Traces 15
1.2 Grid Workload Traces: The Grid Workload Format 22
1.3 Other Interesting Types of Traces 23
1.4 GCT, SWF, GWF, Getting the Best of Each Approach 26

2 Tools for Workload Analysis and Workload Traces Provided **29**
2.1 Xionee 29
2.2 Workload Logs Collection 30

3 Jobs Resource Consumption **41**
3.1 General Context 41
3.2 State of the Art 42
3.3 Resource Consumption Capture 44
3.4 Experiment Environment 47
3.5 Analysis 48
3.6 Discussion and Perspectives 59

4 Clusters Workload Trace Format Proposal **61**

4.1 Which Constraints should we Adopt? 62
4.2 Format Proposal and Conversion 63
4.3 Discussion on the Format Proposed 69

II Experimental Evaluation on Resource and Job Management Systems 71

1 Experimentation on Large Scale Platforms 73
1.1 RJMS Background and Research Challenges 74
1.2 Using Grid'5000 as a RJMS Evaluation Platform 75

2 Reproducibility 79
2.1 Introduction and Motivations 79
2.2 Problem Details and Background 81
2.3 A Tool for Reconstructability: Kameleon 83
2.4 Use Cases 88
2.5 Related Works 93
2.6 Conclusion and Perspectives 94

3 Workload Traces Replay for Experimental Comparison of RJMS 97
3.1 Introduction 97
3.2 Evaluation Process 99
3.3 Experiments 106
3.4 Conclusion & Future Work 115

4 Discussion on Experimental Evaluation 117
4.1 On the Importance of Context Reconstruction for Trace Replay . . . 117
4.2 On the Interest of Replaying a Real Trace 119

5 Conclusions and Future Research Directions 123
5.1 Conclusions 123
5.2 Future Research Directions 124

Bibliography 131

List of Figures

1.1 Trace Collection, Analysis and Data Usage Loop. 10

1.1 Machines Tables. 20
1.2 Job and Task Tables. 20
1.3 Resource Usage Table. 21

2.1 Utilization for the Curie Trace. 32
2.2 Curie Monthly Jobs regarding their Status. 32
2.3 Distribution of Job Core Allocations and Job Lengths for Curie Trace. . 33
2.4 Utilization for the Stampede Trace. 36
2.5 Distribution of Job Core Allocations and Job Lengths for Stampede Trace. 36
2.6 Utilization for the K Supercomputer Trace. 38
2.7 Distribution of Job Core Allocations and Job Lengths for K Supercomputer Trace. 38

3.1 Distribution of the jobs' core utilization means. 52
3.2 Distribution of the mean memory used by job. 52
3.3 Distribution of the maximum memory used per job by an allocated core to this job, values$\leqslant$100%. 52
3.4 Distribution of the maximum memory used per job by an allocated core to this job, values>100%. 52
3.5 Distribution of the jobs' core utilization means. 54
3.6 Distribution of the mean memory used by job. 54
3.7 Distribution of the maximum memory used per job by an allocated core to this job, values$\leqslant$100%. 54
3.8 Distribution of the maximum memory used per job by an allocated core to this job, values>100%. 54
3.9 Distribution of the jobs' core utilization means. 55
3.10 Distribution of the mean memory used by job. 56
3.11 Distribution of the maximum memory used per job by an allocated core to this job. 56
3.12 Distribution of the jobs' core utilization means. 56
3.13 Distribution of the mean memory used by job. 57
3.14 Distribution of the maximum memory used per job by an allocated core to this job. 57

3.15 Size of Files Writes on the DFS on Gofree cluster for all jobs. 58
3.16 Distribution of the aggregated File System Writes sizes (per Minute) on Gofree cluster. 58
3.17 Daily aggregated writes on the DFS. 58
3.18 DFS load (write speed) on Gofree cluster. 58

4.1 SWF Header and its Conversion to Database Format. Note the externalization of Queues, Partitions and Notes and the add of the field *platform_id* to be used as an identifier. 64
4.2 SWF Trace and its Conversion to Database Format. 65
4.3 SWF Extension: Resources Consumptions. 66
4.4 SWF Extensions 68

2.1 Kameleon recipe and steps example. 85
2.2 Environment generation with Kameleon. 85
2.3 Kameleon snapshot feature. 88
2.4 Environment deployment on Grid'5000. 89
2.5 Regular environment customization on Grid'5000. 90
2.6 Environment customization on Grid'5000 with Kameleon. 90
2.7 Environment creation on Grid'5000 with Kameleon. 91
2.8 User Environment aging problem – without Kameleon. 92
2.9 User Environment aging problem – with Kameleon. 93

3.1 Utilization (3.1a) and Queue Size (3.1b) of Curie during the trace reference period. 101
3.2 Trace Context Construction from the Original Trace and Trace Context Reconstruction During the Replay. 104
3.3 Utilization on Curie around the selected trace. The dotted vertical lines delimit the selected trace. No comparison can be made with replayed traces, see paragraph "Limitations of Replay" in Section 3.2. 107
3.4 Queue on Curie around the selected trace. The dotted vertical lines delimit the selected trace. No comparison can be made with replayed traces, see paragraph "Limitations of Replay" in Section 3.2. 108
3.5 OAR utilization and queue size for the 2 schedulers replaying the Curie selected trace. 109
3.6 Slurm utilization and queue size for the 2 configurations replaying the Curie selected trace. 111
3.7 System Utilization for OAR Kamelot and Slurm FavorSmall Configurations 112
3.8 CDF of Wait time for OAR Kamelot and Slurm FavorSmall jobs. Jobs that are not launched have an infinite wait time. 113
3.9 CDF of Slowdown for OAR Kamelot and Slurm FavorSmall. Jobs that are not ended have an infinite slowdown. 114

List of Tables

1.1 Experimental Methodologies Classification as proposed in [61] 8

1.1 SWF Format Description 18

2.1 Curie Inter-arrival Times Deciles 34
2.2 Curie Slowdown Deciles 34
2.3 Stampede Inter-arrival Times Deciles 35
2.4 Stampede Slowdown Deciles 37
2.5 K Supercomputer Inter-arrival Times Deciles 39
2.6 K Supercomputer Slowdown Deciles 39

3.1 Trace Format 46
3.2 Data collected during a measure 46
3.3 Frequent rate overhead 47
3.4 Process number overhead 47
3.5 Clusters characteristics 48
3.6 Trace summary for both clusters. 48

3.1 Utilization, jobs launched and jobs completed for different configurations. 112

4.1 Utilization, jobs launched and jobs completed for the two best configurations with and without workload context reconstruction. 118

Chapter 1

Introduction

High Performance Computing (HPC) is characterized by the rapid, constant growing of architectures sizes. With the increase of the computing systems' resources, grew the users needs for computing power. High Performance Computing is now preparing the era of the transition from Petascale to Exascale. Distributed computing systems are already facing new scalability problems due to the increasing number of computing resources to manage. It is thus necessary to study in depth these systems and comprehend their behaviors, strengths and weaknesses to better build the next generation.

In many different fields, HPC infrastructures have become a major tool for research or production problems. Astrophysics, Climate, Geo-Sciences, Movies, Medicine; all of them need nowadays' parallel distributed systems computing power to drive their simulations, experiments and computations. With a large panel of distributed systems available with different abstraction level, services and constraints such as: clusters, grids, clouds; all the application domains can now benefit from the best type of platform regarding their usage and needs.

Classic supercomputing applications that need a tight coupling of the computing resources such as fluid dynamic simulations, tend to prefer the cluster approach, which guarantees a higher performance and reliability in particular for the networking and I/O. In cluster computing approach, the computing resources tend to be homogeneous[1], tightly coupled and connected through a high speed interconnect. Another characteristic is that they are linked to a common Distributed File System (DFS) that serves users files on all the nodes of the infrastructure. In these kind of systems, a central component called the Resource and Job Management System (RJMS) is in charge of managing the users' tasks (jobs) on the system's computing resources. The functionalities of such tools are:

- request for a set of resources by the user to run his tasks. This process of resource request is generally called "submission" while the users set of tasks is called "job". We thus talk about job submission when a user requests resources to the system to run his tasks.

[1]heterogeneous clusters exist but are mostly experimental

- Scheduling of the user jobs according to all their resource constraints, administration and usage rules like Quality Of Service (QOS) or fairness.
- Launching and cleaning of the user jobs on the allocated resources.
- Management of the resources along the job's life.
- Connection control between the resources involved in a given job.
- Accounting and resources utilization control.

These assignments can be grouped in three principal abstraction layers: the declaration of a job to the system, the scheduling of the jobs upon the resources and the job execution and resource management control. Thus the work of a RJMS can be decomposed into three main, interconnected subsystems: Job Management, Scheduling and Resource Management.

The RJMS is thus a central component of a cluster infrastructure as it stands between the platform hardware and software, the users and their jobs, the administrators and their goals. With more and more resources to manage and with the hardware constant evolution, these systems have to deal with new challenges. Scalability, energy, new types of resources as the recent apparition of Graphics Processing Unit (GPU) and General-Purpose Graphics Processing Unit (GPGPU) in supercomputing, push RJMS to quickly adapt to their environment. This has conducted to a lot of research works on the study of these tools, always more complex and powerful, that follow the evolution observed in hardware components.

Recent advances in hardware technology and integration have pushed the raw computing power of machines from Gigaflops to Petaflops: 10^{15} FLoating point Operations Per Second (FLOPS). Famous examples of such systems are the 1st supercomputing machine in the 41st edition of the TOP500[2] list: Tianhe-2, with more than 33 Petaflops, followed by Titan, a Cray XK7 system with 17.59 Petaflops and Sequoia, an IBM BlueGene/Q system with 17.17 Petaflops. Nowadays' most powerful machines have a number of computing cores on the order of the million. The evolution of more and more complex network interconnects also takes part of the global efficiency of these machines as they provide high speed communications between the computing resources and allow them to share and transmit data as fast as possible.

The performance of an HPC system is obviously determined by the unitary performance of the subsystems that compose it, but also by the efficiency of their interactions.

1.1 Motivation

The research on High Performance Computing faces up challenges from all the different layers of computer science. The hardware, constantly evolving, the soft-

[2] http://www.top500.org

ware applications that have larger and larger resource requirements and the RJMS middleware that has to manage the resources and serve the applications. These three principal sub-systems have a level of performance that needs to be continuously pushed to the top. In the quest for performance, each of these sub-systems is improved and tuned by specialists independently from the others because of the difficulty to master the whole stack in all its details. Thus, research in performance evaluation in HPC is mainly focused on one of these sub-systems. Few research works have been conducted that mixed several of these layers or try to evaluate HPC systems in their globality.

In [54], the performance of several RJMS was tested at scale on a real platform and not simply in simulation. This approach used a RJMS benchmark whose underlying workload model is known to be perfectible [47]. However, despite the fact that the model used was rough, it is the only work that considered the evaluation of a RJMS, deployed at large scale on a real platform, with a concrete experimental point of view. This enabled to test in real case scenarios different RJMS features along with scalability evaluation.

The research that is made through this thesis retakes the originality of the experimental approach mentioned earlier but in a context that seeks to be as close to reality as possible. This implies to rethink the evaluation method and to use a workload source that can reflect a real system behavior. The question this thesis wants to answer is thus: *How can we find workload sources that reflect real system behavior and how can we use these sources in experiments for performance evaluation of HPC systems?*

1.2 Distributed Systems Performance Evaluation

To evaluate a distributed system performance, we generally give a grade to a given system and compare it with its peers. This is later used to categorize the system. One common evaluation method is to classify the computing machines regarding their raw computing score at the HPL[3] version of the Linpack[26] benchmark, used to rank supercomputers for the TOP500 list. The Linpack makes use of Message Passing Interface (MPI) communications, the Basic Linear Algebra Subprograms (BLAS) or the Vector Signal Image Processing Library (VSIPL) libraries to solve dense linear arithmetic problems that require a lot of computing power. Thus, this benchmark provides a method to quantify the time taken to solve the same problem by different systems and the accuracy of the solution they found. This enables to classify them and to attribute them a raw computing power score given in FLOPS.

Although this raw computing power is an important metric of the system's performance, it is not the sole. When running the Linpack for the cluster qualification, the full system is dedicated to this benchmark. In production mode, the computing power is shared between several users and the management and orchestration of their

[3] http://www.netlib.org/benchmark/hpl/

jobs has to be done. Thus the software and middleware part also have an important role on the global performance as:

- the applications that use these systems need to be able to scale up to this high number of resources to use efficiently their computing power,
- the middleware management and in particular the RJMS has to be able to efficiently deliver in a fair way this computing power to the applications.

In this thesis we focus on the performance of the RJMS and the harmony of its configuration with the users' workload. RJMS middleware has to deal with several problematics to be efficient. First, it has to be able to equitably distribute the computing resources to user applications. This is one of the prerequisites for user satisfaction. Then, it has to keep a fairly high level of utilization of the platform and avoid utilization "holes" as much as possible. This for two reasons: if the low utilization comes from a bad management of the resources, this impacts the users (and administrators) satisfaction as they can see that the platform is not fully used while the jobs are still waiting to be granted for computing resources. If the low utilization is caused by an intrinsically low workload by the users, administrators should wonder if that is a sign that the design of the platform was wrongly over-sized. This is not always the case as a platform can be designed explicitly to support an average load much lower than shorter high peak loads. This may lead to the use of energy saving techniques that idle unused resources to save electrical power.

As discussed in [93], a high utilization of the platform often involves a higher slowdown for the jobs (longer system response to their request). Thus, the RJMS has to find a good trade-off between utilization and slowdown. Still in the same work, it was found that on data analyzed, an acceptable slowdown associated with utilization was found to be near 60%. This, of course, is valid for the workloads analyzed in earlier mentioned study and thus may differ from other types of workloads.

Thus, with the interaction of many components and the complexities of the underlying workload, RJMS evaluation in a global point of view is difficult. What is generally done is, given a set of performance metrics, look whether their values seem fair or not.

Evaluation Metrics

In [93], five general metrics are proposed to evaluate the platform usefulness to the users and its Return on Investment (ROI). These are generally the metrics that are taken as specification constraints at the time of system design. They are later used to determine if the system actually meets its initial expectations.

- *Allocation.* This criterion rates the agility of the system to meet user needs and expectations in terms of resource requests. It is the system ability to provide to the users, the resources they need. e.g. the users may need a resource allocation at the node level or at a finer grain level, like cpu core. In this later case, the

system should be able to provide to the users a method to ask for cpu cores instead of full nodes.

- *Availability.* Generally this metric is the fraction of the time the system is available to serve user requests. More particularly for [93] in computing systems evaluation, this criterion gives the capability of the system to schedule quickly high priority jobs and to schedule in a reasonable time the other jobs. This can be depending on the number of resources itself (the capability criterion) and/or by the scheduling and priorities techniques adopted.

- *Capability.* This criterion tells if the system is well dimensioned to fit the demand for the jobs that request the most resources (*i.e.* large jobs).

- *Response Time.* Before explaining how this criterion is used for system valuation we need to give a more precise definition of response time as it is not used in the same way depending on the application domain. In [68](p.37) R Jain gives two general definitions of response time. It can be either the interval between the end of the user request and the beginning of the system response (definition 1) or the interval between the end of the user request and the end of the system response (definition 2). In our context user requests are job submissions. Still according to [68], the second definition is preferable if the time to process the response is long, which is the case. R Jain also proposes for a batch stream the notion of *turnaround time* corresponding to the interval between the submission of a batch job and the completion of its output. Thus, in the context of large scale distributed systems we define response time (equivalent to turnaround time) as in definition 2. The time between the submission of a job and the beginning of its execution is called *reaction time* (or *wait time i.e.* the time the job waits in the queue). The analysis of response time tells if an application benefits correctly of the use of several resources in parallel. Response time of a parallel run of an application should be always shorter than its sequential execution time.

- *Throughput.* It is the rate at which the requests are served by the system. In the context of batch streams, the throughput is measured in jobs per second. The value of this metric is chosen at the specification time of the system design.

The Response Time metric, however is also dependent on the workload and system load. Used as a specification, it will define what should be the amount of time a job can stay in the system. Used as a performance metric it gives us information on how much a job had to wait before being able to run on the platform.

Along with these design and ROI metrics, several major performance metrics are commonly used for the evaluation of distributed systems that already are in production mode [93, 68, 116, 54, 44, 76, 120]. These metrics give a valuation of a system configuration and its response regarding the workload of the users.

- *Utilization*. Generally this metric measures the fraction of time the system is busy servicing requests. In our context one accepted definition is the ratio of the computing resources allocated to the jobs over the available resources in the system.
- *Slowdown* (or *stretch factor*) is the response time of a job at a given load over its response time at the minimum load.
- Other metrics based on the reaction time or response time and weighted or normalized by the job execution time like *average weighted wait time* and *average weighted response time* as used in [37] or by a lower bound like *bounded slowdown* in [44].

While the first ones are ROI metrics, the later are more qualitative of the capacity of the system to process users load. More dependent on the system context at a given time, they give us a deeper view of the system performance. System context can be decomposed into

- the platform context: the RJMS, its configuration and administration,
- the usage context: the users, their workloads.

Thus when we evaluate a platform with Utilization, Slowdown and Wait Time based metrics, we evaluate the couple RJMS - workload. It is not guaranteed that the same system with a different workload would lead to the same performance measures.

This has motivated the study of the HPC platform workloads and the creation of several models to deeply study the interactions between system and workload.

The Parallel Workload Archive (PWA)[4] proposes a collection of different models for that purpose. The Calzarossa and Serazzi (in 1985) [12] model proposes a model of the arrival process of interactive jobs in a multiuser environment with respect to the daily cycle. Leland and Ott (in 1986) [81] and Downey (in 1997) [29] propose models of jobs runtimes. Then, more complete models followed [38, 70, 44, 82, 20] that provide information on the distribution of arrivals and runtimes. The most complex of them also made a distinction between rigid and moldable jobs, impact of the daily cycle or even correlations between the runtime and the number of resources in the system.

Lately, Tsafrir (in 2005) [112] provided a model of user runtime estimates. These estimates by the users are well-known to be wrong [6, 112, 80] and are used in Backfilling strategies of modern RJMS which are studied in several works [85, 78, 79].

All these models have been presented and used in many works [44, 53, 48, 40, 27, 19, 42, 43, 28, 46, 13] on workload modeling, system evaluations or comparisons and scheduling algorithm evaluations. These works used the aforementioned performance metrics for their evaluations.

For more information, the PWA lists, classifies and gives pointers to these models in a dedicated page:

[4] http://www.cs.huji.ac.il/labs/parallel/workload/models.html

`http://www.cs.huji.ac.il/labs/parallel/workload/models.html`.

The classification is made regarding the model features and give a brief description of each model and lists the works were they have been used.

A discussion on the construction of a model based on the observation and the statistical study of workloads characteristics is also proposed in [41].

The necessity to study systems performance and the design of complex models lead to the interest on the study of workload traces of production systems. In the next section we discuss on the analysis of such traces.

1.3 Workload Traces Analysis

The concept of **workload** has several meanings depending on the context where it is used. If we talk about an application, the workload is the load induced by the application to the system (the sequence of processes and system calls). If we deal with a machine workload, it is the load induced by both the system itself and the applications running on the machine. For distributed systems, the workload refers to the set of all individual jobs that are processed by the system.

The complexity of managing user applications on the resources conducted to the analysis of the workload the platform has to support. The analysis of the workload is a critical step in distributed systems performance evaluation and optimization. Understanding how the users exploit and share the platform enables us to adapt the system's response to these usages and to provide them a more efficient service.

The need for workload comprehension has led to the collection of traces from production systems and to the proposal of a standard workload format. These contributions enabled the study of numerous of these traces. This also led to the construction of several models, based on the statistical analysis of the different workloads from the collection.

Until recently, existing workload traces did not enabled researchers to study the consumption of resources by the jobs in a temporal way. This is now changing with the need for characterization of jobs consumption patterns. With the venue of Big Data, the study of these patterns has became necessary to prepare the systems to a high load and volume of data to manage. The knowledge of jobs consumptions patterns will help us having the good decisions in future generation platforms design.

In the first part of this thesis we propose a study of existing workload traces. Then we propose an observation of cluster workloads with the consideration of the jobs resource consumptions over time. This highlights specific patterns of usage of resources from users and conducts to scheduling propositions. Finally, we propose an extension of the former standard workload format that enables to add such temporal consumptions without loosing the benefit of the existing works based on this format.

1.4 Experimental Evaluation

In [61], Gustedt et al. propose a survey of experimentation methodologies for large scale systems. They classify these methodologies according to the type of environment in which the experiment is driven and the type of application used as input. This classification is illustrated in Table 1.1.

Environment / Application	**Real**	**Model**
Real	In-Situ	Emulation
Model	Benchmarking	Simulation

Table 1.1: Experimental Methodologies Classification as proposed in [61]

In [54], Georgiou discusses the benefits and drawbacks of the different classes of experimental methodologies in the context of distributed systems performance evaluation.

Experimental approaches based on workload models have served the goal of distributed systems evaluation. Existing models used for that purpose describe the average behavior of observed systems. However, although the study of average behaviors is essential for the understanding of distributed systems, the study of critical cases and particular scenarios is also necessary. The study of critical cases would give a more complete view and understanding of the performance of resource and job management. This also enables to detect potential problems and bottlenecks either in the job scheduling or resource management part of the RJMS. A simple example is the use of a very dense workload with short inter-arrivals to test the ability of the RJMS to bear high job submission throughputs. As the trace comes from a real workload, we know that this throughput is realistic, at least in the platform from where the trace is taken. In the second part of this thesis we propose an experimental method for performance evaluation of distributed systems, based on the replay of production workload trace extracts. These extracts, replaced in their original context, enable to experiment the change of configuration of the system in an online workload and observe the different configurations results. In this approach, the replay of the workload trace is finely set-up and the reconstruction of its context enables to mimic as much as possible the original platform.

1.5 Evaluation Process

Experimental evaluation of distributed systems is a complex task. Whether it is conducted through model based or trace extract based experimentations, the important step is the understanding of the underlying workload that will be used in

the experimentation. When we replay a workload[5] on a given platform[6], we have to ensure that what we replay is what would have been naturally run in the platform in a real-life case. A model, even good enough can lead to unreliable results if its input parameters distort too much the natural tendency of the workload. A workload trace, extracted from a totally different type of platform might have a low value (e.g. if we replay a workload of a platform of size 2^N on a platform of size 2^{N+1} the load on the system is divided by 2). Thus, what should be done is the understanding of the full cycle of workload trace extraction, analysis and experimentation, as illustrated in Figure 1.1. The first step to master is thus the capability of collecting interesting information from the workload of an existing production system. This data can be retrieved from the logs of the nodes or from the RJMS. A step of data correlations can be necessary if data comes from multiple sources. Then, we end up with a usable workload trace. The second step is the understanding of the workload trace. By statistical analysis, graph observations, metric analysis and comparisons to other system's traces, we have to try to understand as much as possible what the workload has to say. From this analysis, a workload model can be derived; or, according to the more experimental approach, a trace extract can be chosen. The third step is the experimentation. Based on simulation, emulation, benchmarking or in-situ experiment, this part of the evaluation process will enable to test experimentally an other system, or the same system with a different configuration. By using either the model or workload traces in the experimentation, we try to mimic the original system behavior during the experimentation. By this mean, we highlight the impact of the changes on the workload and its performance metrics. This experimental step will be the last one of the first round of the cycle, we have to analyze data from the experiment that was just driven and start all over again the cycle of data extraction and analysis (and possibly continue with experimentation).

1.6 Contributions

This research work led to several contributions detailed in the following sections.

Workload Analysis

In the field of workload analysis, the contributions of this work are:

- the proposal of a set of tools for workload analysis, the production and distribution of several workload traces from large clusters along with their respective analysis,
- the analysis of two clusters workload traces with the consideration of resource consumption by the jobs [35, 36],

[5] from a model instance or real trace

[6] whether it be a real production platform or a reconstruction of this platform dedicated to the experiment

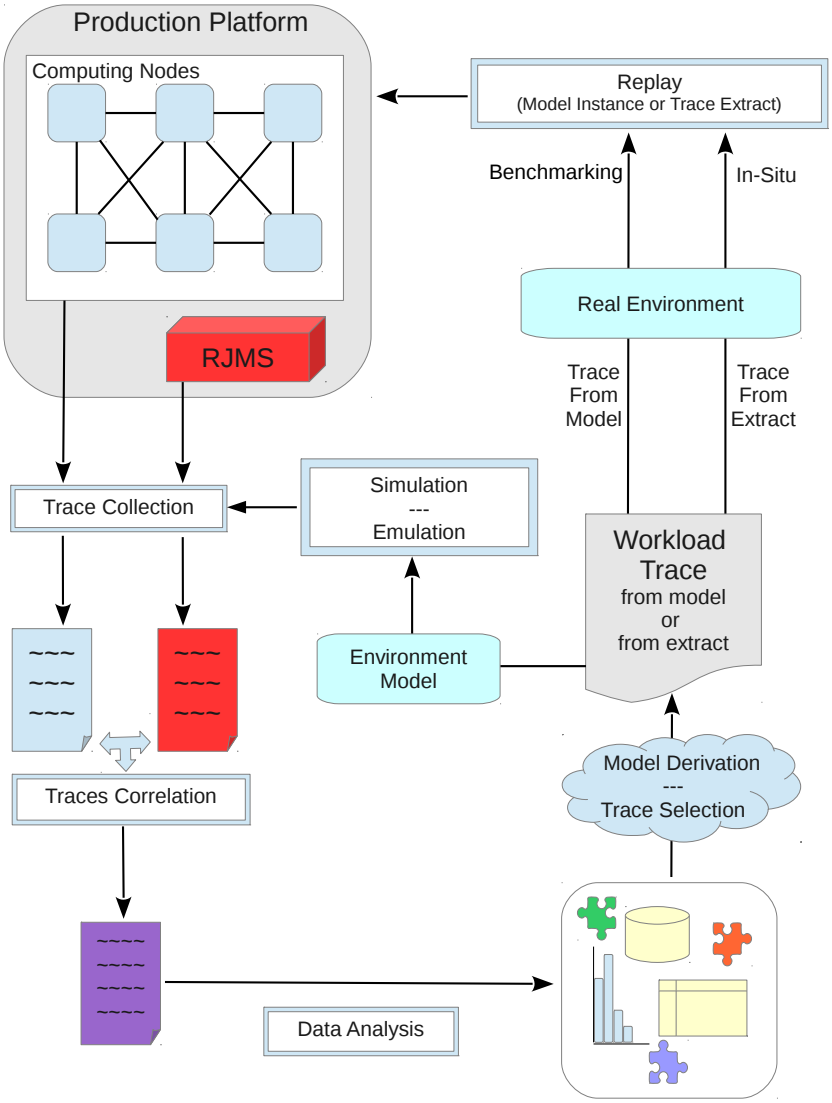


Figure 1.1: Trace Collection, Analysis and Data Usage Loop.

- the proposal of a new workload trace format.

Trace Replay

A second contribution of this work is the proposal of an experimental methodology and a tool for workload trace replay, validated on a real case. This methodology enables to evaluate the RJMS performance regarding a given scenario and to detect potential problems.

Reproducibility

A third contribution is the proposal of a tool that aims at managing experiment environment [34] and which is used in the experimental methodology of trace replay.

1.7 Thesis Overview

The organization of the thesis is as follows.

In Part I we first present our works in workload traces analysis. In Chapter 1 we review existing workload trace formats and state their advantages and drawbacks. In Chapter 2 we propose a set of tools to ease workload analysis and we provide several workload traces from large production platforms. In Chapter 3 we propose an analysis of two workload traces that take into account the consumptions of the resources by the jobs. Finally, in Chapter 4 we propose a new workload format that synthesizes the advantages of the different approaches of existing formats and that enables to include resource consumptions in the workload trace.

In Part II we present our works in experimental evaluation on RJMS based on workload traces. In Chapter 1 we present experimentation methodologies on large scale platforms. In Chapter 2 we present a tool that will be used in the experimentations and whose goal is the management of the experiment environment. In Chapter 3 we propose our evaluation methodology of RJMS based on workload traces replay, put back in their original context. In Chapter 4 we discuss the methodology proposed and give some future works.

Finally, we conclude with a discussion on the use of the full analysis – experimentation cycle for distributed systems evaluation and optimization and give future research directions.

Part I

Workload Traces Analysis

Chapter 1

State of the Art

In this part we focus on the analysis of workload traces as a mean to study and optimize clusters' behavior and performance.
The first question to ask when dealing with this subject is *what should be a workload trace?*

Several workload trace formats exist in the literature. They are not always the consequence of a research project, most of them come from ad-hoc formats provided by particular systems as logs for administrators. These do not aim at specifying a standard format. However they may have an interesting vision of what is a workload trace and this is why they worth being looked at. In this section we have an overview of some of the existing workload trace formats that can be used in our context. This overview does not aim to be exhaustive but to give an idea of their similarities and differences. We make the distinction amongst the different formats depending on the kind of architecture they aim at.

1.1 Cluster Workload Traces

Workload Traces Provided by RJMS

Most of the widely used RJMS provide an ad-hoc workload format to report to the users and administrators the characteristics of a set of jobs. Generally this reporting is done through a command, provided by the system that gives access to these characteristics. Typically, one type of information provided is related to the jobs' scheduling and arrival, such as the time a given job ran, when it was submitted, how long it stayed in the queue, the set of resources that were allocated to this job. An other type of information provided is about the job request to the system, like the queue or partition used to submit the job, its requested time and the resource constraints (in terms of nodes, cores, memory or in some cases more specific physical or software constraints on the requested nodes). User characteristics are also provided like the name and the group of the user that submitted the job. Most of RJMS provide these basic information. For more information on a particular batch system, a

survey of current RJMS software and their common and specific features is available in [54].

In addition to these basic information some RJMS provide data on the jobs resource consumptions. It is the case for SLURM[119][1] and the MOAB[2] suite with the scheduler MAUI[67][3] that automatically monitors the resources usage during job execution. Recently, Slurm became able to profile user jobs consumptions and store collected data in HDF5 files[4]. Collected data are: energy, Lustre and network accesses along with common tasks metrics (cpu, memory, I/O). The OAR[14][5] RJMS provides this kind of information on request, before the submission of the job.

Typical information provided is cpu time, amount of memory (mean and maximum), network bytes sent and received by the job. This kind of information is an accumulated value over all the processes on all the nodes allocated to the job. MAUI has an interesting property not proposed by the other systems, it also provides resource traces along with job traces. These traces enable to retrieve the nodes states and configurations, which is particularly useful to retrieve the system state at a given date.

To summarize, several ad-hoc workload traces exist and provide different type of information. Amongst them, some seem very useful to include in what should be a workload trace. The basics jobs characteristics (arrival, start, stop, user), jobs requests (cores, memory, queue or partition, time) are obviously needed. However other important characteristics such as jobs resource consumptions and node failures should also be included in the workload trace. Jobs resource consumption is interesting because it enables to characterize a given job and its impact on the physical resources of the system. Nodes failures is interesting because it enables us to understand job failures and sub-utilization of the cluster. It is also needed to understand the scheduler decisions in a post-mortem analysis. As the RJMS knows the system state, its scheduling decisions are determined by this state.

Standard Workload Format

The Standard Workload Format (SWF)[16] is an initiative to make workload data on parallel machines freely available and presented in a standard format. This work, along with workload data collection, are presented in the Parallel Workload Archive (PWA)[6]. The idea of the PWA is to collect and to redistribute several traces from real production systems built from the logs of computing clusters. To achieve this goal the authors have defined the SWF format that aims to be a standard and is designed

[1] `https://computing.llnl.gov/linux/slurm/sacct.html`

[2] `http://docs.adaptivecomputing.com/mwm/Content/topics/analyzing/workloadtrace.html`

[3] `http://docs.adaptivecomputing.com/maui/trace.php`

[4] `http://slurm.schedmd.com/hdf5_profile_user_guide.html`

[5] `http://oar.imag.fr//sources/2.5/docs/documentation/OAR-DOCUMENTATION-ADMIN/#oarmonitor`, last accessed July 2013

[6] PWA: `http://www.cs.huji.ac.il/labs/parallel/workload`

for an easy use of workload logs. With SWF one can work with several logs with the same tools and the format enables to abstract the complexity of mastering different ad-hoc logs. It is a big step in the field of workload analysis. Many other works were based on this format, in particular for workload models generation. Following the same goal as the ad-hoc RJMS logs, SWF gives information about the job requests, allocations, characteristics and resource consumptions.

Each workload is stored in a single ASCII file which is composed of two parts. First the header comments which gives information on the platform, its configuration, the scheduling queues and other trace characteristics. This metadata information is very important for the understanding of the trace it refers to. In particular the details on the queues and partitions configurations help in the analysis of job data. Depending on the underlying RJMS and its configuration, submission queues may have different properties such as priority, maximum run time for the jobs or maximum number of cores that can be allocated. If we consider the trace data in itself without considering its metadata, we miss a very important information. This can drive us to a wrong comprehension of what happened in the trace and why. More generally, the more information we have on the environment of the trace data, the more our vision of the data will be correct and our analysis valuable.

The second part is the workload trace itself where each job is represented by a single line in the file. For each job is given temporal and scheduling information like its submission time, runtime, the time spent in the waiting queue; user request characteristics such as the number of processors, the amount of memory and time requested by the job, the queue and/or partition where this job was submitted and the user that submitted the job and his group. SWF gives also information about global job resource consumptions. These consumptions are averages for the cpu and memory resources. For the cpu resource, it is the average over all processors of the CPU time used. For the memory resource, it is the average of memory used per processor. A complete description of the format fields is given in Table 1.1 and fully detailed in the SWF description page [7].

The particularity of this format is that anonymity is guaranteed intrinsically as every field is numeric, even for the user id and their associated groups which are mapped by integers regarding their order of appearance.

SWF is simple to handle and contains all the information on the basics jobs characteristics and requests. It also gives some information on their average consumptions of cpu and memory. To this, it adds several valuable information as the final status of the job (error, completed or canceled) and the notion of precedences between jobs. The field *Preceding Job Number* enables to know on which job was dependent the current job, i.e. the current job can only start after its dependent job termination. An other contribution is the *Think Time* information, which is the time the user took after a job termination, to think about the next job submission. Think-times are used in several works [5, 4] on user modeling and session detection. This enables to give an information on user feedback on the workload.

[7]SWF: `http://www.cs.huji.ac.il/labs/parallel/workload/swf.html`

SWF Format	
Headers comments	Fields
1. Version	1. Job Number
2. Computer	2. Submit Time
3. Installation	3. Wait Time
4. Acknowledge	4. Run Time
5. Information	5. Number of Allocated Processors
6. Conversion	6. Average CPU Time Used
7. MaxJobs	7. Used Memory
8. MaxRecords	8. Requested Number of Processors
9. Preemption	9. Requested Time
10. UnixStartTime	10. Requested Memory
11. TimeZone	11. Status
12. TimeZoneString	12. User ID
13. StartTime	13. Group ID
14. Endtime	14. Executable (Application) Number
15. MaxNodes	15. Queue Number
16. MaxProcs	16. Partition Number
17. MaxRuntime	17. Preceding Job Number
18. MaxMemory	18. Think Time from Preceding Job
19. AllowOveruse	
20. MaxQueues	
21. Queues (list)	
22. Queue (description)	
23. MaxPartitions	
24. Partitions (list)	
25. Partition (description)	
26. Note	

Table 1.1: SWF Format Description

However SWF misses information on node failures and jobs resource consumptions. The consumptions provided by SWF are the average cpu time used and average memory used per processor, this is not really enough. To give a better information on the jobs physical impact on the nodes, the information on maximum memory used per processor should also be included along with other resource consumptions such as disk and network.

Google Cluster-usage Traces

Google Cluster-usage Traces (GCT) was first announced in [90]. A second work [91] was later published to explain the choices made to clean and anonymize the trace for public release. This trace is available for downloading on the Google Cluster Data

Wiki [8].

GCT is a workload trace format composed of several files, where each file contains a particular information about a specific part of the workload. e.g. a specific file is dedicated to job "events" and describes the arrival, start and end times of jobs; an other file gives information on the jobs resource usage. The files are provided in CSV format (list of values separated by commas). Although the authors of GCT propose this format as plain text, it has many similarities with a relational database. Even in the presentation of the format[9] itself, the authors present the "Tables", their "keys" and "fields" which belongs to database vocabulary. When we observe more attentively GCT, it is remarkable that it is in fact a relational database, but implemented as a set of CSV files. Each file is a database table and each file line would correspond to a table row. Common fields in the files can be used to link them together, as it would be done in a relational database with common attributes. This choice of a database format but implemented as plain text is not really justified but is interesting. As they provide data compressed and stored in several sub-files, an explanation of this choice might be the ease of redistribution of the data and the storage space saving. A set of compressed files is smaller, in terms of disk space, than a full database dump.

GCT is more complete than SWF on several aspects. It embeds information about machine events as proposed in [74] along with machine attributes that can be specified as a constraint by the jobs. The file formats describing machine events and attributes are presented as database tables for a better understanding in Figure 1.1. The table *machine_events* gives information about each machine in the cluster. The field *event_type* is used to tell the type of the event. Three values are possible. *ADD*: for the arrival of a machine in the cluster (this can be a new machine or an already existing machine that became available again), at least one event of this type is present for each machine. *REMOVE*: when a machine is removed from the cluster, this can be a definite removal or a temporary removal because of a failure or a maintenance, in this case when the temporary event is finished, a new *ADD* event will occur to set the machine as available again. *UPDATE*: this field indicates that a machine was updated in terms of cpu or memory, in this case the field *capacity_cpu* and/or *capacity_memory* will give the updated value. As such an event is also characterized by a machine ID and a timestamp, it is easy to reconstruct the cluster status along time (nodes add and failures).

The table *machine_attributes* gives information about the attributes such as kernel version, clock-speed, presence of an external IP address. These attributes are key-value pairs. The field *attribute_name* is associated to the field *attribute_value* to define these attributes. Because of the key-value pair structure, there is no restriction on the type of the attributes. They are obfuscated and provided either as integers or obscured strings. These attributes will also be used as constraint in the table *task_constraints*. As machine events, machine attributes are for a given machine at a given time, which enable to retrieve the nodes hardware upgrades.

[8] `https://code.google.com/p/googleclusterdata/wiki/ClusterData2011_1`

[9] in a document linked here: `https://code.google.com/p/googleclusterdata/wiki/`

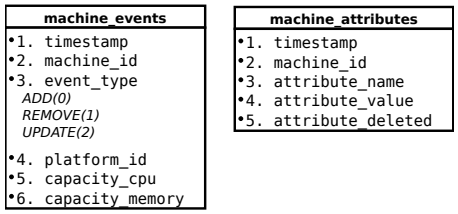


Figure 1.1: Machines Tables.

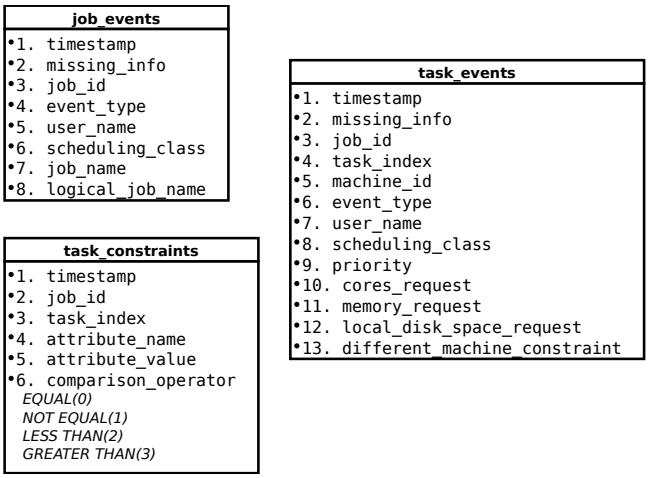


Figure 1.2: Job and Task Tables.

Along with the description of the machines, GCT also provides information about the job and task characteristics and constraints. Related file formats are presented in Figure 1.2.
The tables *job_events* and *task_events* have a great similarity with SWF. They give information about the jobs arrival time, wait time, run time, user, cores and memory request. One difference however is that in SWF, submission, start and end dates of the jobs are fields in the job information row. In GCT it is the couple <*event_type , timestamp*> that gives these times. The field *event_type* tells if the associated timestamp concerns the arrival, start or end of the job. Thus in GCT, for each job that ran normally, 3 rows will be inserted in the table *job_events* for each of the 3 events. This has the drawback of creating a lot of data redundancy as the other fields in the table (*job_id, user_name, job_name*) will be the same for the 3 rows, as they depend on the job characteristics and not its events. Moreover, the same field *event_type* is also used to give what SWF calls the status of the job. i.e. if the job ended normally, was canceled or terminated in error. The way how the *event_type* field is used in GCT is confusing and contributes also to data redundancy.

As opposed to SWF, in GCT the jobs are divided into tasks, each of them running on a machine. Thus, the field *machine_id* in the table *task_events* enables to retrieve the list of the machines allocated to the job, which is an interesting information to correlate with the failures data table in order to know if a given job had to suffer from a node failure. The table *task_constraints* may also give valuable information about additional constraint other than the number of cores and the amount of memory. This information is important to better understand the scheduling decisions made at the time of the job submission. These constraints on the tasks refer to the machine attributes given in the table *machine_attributes*. This kind of information is not supported by SWF.

Figure 1.3 presents the table of resources usage by the job tasks. For each task of

---

`TraceVersion2`

resource_usage
•1. start_time
•2. end_time
•3. job_id
•4. task_index
•5. machine_id
•6. cpu_usage_mean_and_max
•7. memory_usage
•8. assigned_memory
•9. unmapped_page_cache_memory_usage
•10. page_cache_memory_usage
•11. maximum_memory_usage
•12. disk_io_time_mean
•13. local_disk_space_used_mean
•14. cpu_rate_max
•15. disk_io_time_max
•16. cycles_per_instruction
•17. memory_accesses_per_instruction
•18. sampling_rate
•19. aggregation_type

Figure 1.3: Resource Usage Table.

each job, this table gives information about the task consumption for the following resources:

- cpu,
- memory used, assigned, cached and maximum,
- local disk IO time (mean and max) and space used.

Each row gives the resource consumptions of a task between two timestamps (fields 1 and 2), this interval is called "measurement period". It is also given what was the sampling rate between these two timestamps and what is the type of aggregation used to report the values. Usually, a measurement period is 300 seconds, however this interval may be shorter when tasks are updated within this period. In this case, the measurement period will be shortened and a new one of 300 seconds will start. Usually the sampling rate within a measurement period is 1 measure per second. In case of high system load, this rate might be lower and this justifies the presence of the field *sampling_rate* that provides this information.

About the values reported in a measurement period, for each metric they are generally the averages of each sample within the period and the maximum of the sample values. However for some kinds of resources, the maximum has no meaning and what is reported is actually the sum of the maximums. In this case, the field *aggregation_type* will be set to 1 (0 being the normal case).

The approach proposed here is very interesting because it allows to retrieve jobs consumptions in a temporal way. With the values given for each measurement periods we can reconstruct what was the job activity on the resources and this enables jobs characterization. With this format, not only the averages but also the maximums values are provided and this, in particular for the memory resource, is a valuable information.

To summarize, the approach proposed by Google in the GCT format is very interesting. Its database-like format allows to combine many information from different sources as jobs characteristics, requests, consumptions, nodes failures and configurations. However the presence of data redundancy, the complexity to process data from the different files (they do not provide the database format itself that would have helped), makes it perfectible and also more difficult to use than SWF. Moreover, as Google clusters have a different usage and workload than common HPC platforms, several information are absent from GCT and would miss for a classical HPC cluster, such as submission queue, user requested time (walltime) and jobs dependencies. The user requested time information is very important if the scheduling uses backfilling (which is commonly the case in clusters), as this determines if a job has enough room to be backfilled or not.

1.2 Grid Workload Traces: The Grid Workload Format

The Grid Workload Format (GWF)[66] provided in the Grid Workload Archive (GWA)[10] is an extension of SWF to support grid oriented traces. The goal is to enable the use of grid workload traces in a unitary format as does SWF for cluster traces. It is worth to be noted that this format is database friendly and that workload data available in the GWA is provided in two flavors: plain text and SQLite[11] database. It adds some grid specific aspects like sites, virtual organizations and workflows. Workflows supported by GWF are Bag of Tasks (BoT), Direct Acyclic Graph (DAG) and Direct Cyclic Graph (DCG) which are common methods to group sets of jobs in a grid environment. Other types of workflows may be described as well with the key pair *"ExtensionID":"Params"* where *ExtensionID* is a workflow model described in the GWA and *Params* the parameters for this workflow model.

The format embeds information about network and local disk space usage and redefines the way how the other resource consumptions are described. A list of comma separated pair *"Description of the Resource":"Consumption"* enables to describe the resources consumptions at will. As for workflows description, the description of the resource have to be registered in the GWA. If this description is not available in the archive, it is also possible to embed it in the trace itself.

GWF proposes an extended description of the resources requested. The CPU Architecture, the OS and its version, the type of network and the local disk space requested are directly provided as fields in the format. The other resources requested are described with a string. In an extension to the format, GWF also provides information about advance reservations and the origin of failures (infrastructure, middleware, application or user cause). Advance reservations, i.e. the reservation of

[10]GWA: `http://gwa.ewi.tudelft.nl/pmwiki/`
[11]http://www.sqlite.org/

resources for a given date, is a very important information to add, as these deeply impact the scheduling decisions [83, 103, 104].

For more information, the authors provide a document[12] describing GWF and its similarities with SWF.

To summarize, GWF proposes several enhancements to SWF as the notion of advance reservations, the capacity of extending resources usage and request description and a better information on the origin of the job failure. The idea of providing both a plain text along with a database format is interesting, as it enables to benefit from the power of database tools. However, GWF is not as complete as GCT format as it does not allow to describe node failures nor temporal jobs resource consumptions.

1.3 Other Interesting Types of Traces

In this section we review some other interesting trace formats that are not workload traces per-se but that might worth to be included in a workload trace. They are machine traces and failure traces that could give interesting information on the cluster nodes states and I/O traces that could be interesting to report in jobs consumptions.

Machine Traces

We first present two tools that collect information on the nodes status, usage and load. The first one is dedicated to the Grid'5000 platform and allows users to redefine their useful metrics. The second one is an OpenSource tool that uses a time series approach to collect data at a high rate.

- **The Grid'5000 Metrology API**: The Grid'5000 research platform provides an API to retrieve information on the values of system counters[13]. Based on the Ganglia[14] tool, the Metrology API collects a predefined set of metrics that are registered for each node, for instance memory consumption, cpu consumption, bytes in, bytes out. The real interest of the approach however is the capability provided by this system for users to redefine their own metrics as the number of requests per seconds for an HTTP server, or any other kind of experiment-specific metric. The Metrology API collects and store metrics data for each node along time. This requires that user jobs owns the full nodes in their reservation, to guarantee to not be disturbed by other job activities and thus to ensure the validity of the metric values.

- **The Time Series Approach of OpenTSDB**: OpenTSDB[15] is a distributed, scalable Time Series Database (TSDB) written to store, index and serve metrics

[12] `http://gwa.ewi.tudelft.nl/TheGridWorkloadFormat_v001.pdf` **last accessed July 2013**
[13] `https://www.grid5000.fr/mediawiki/index.php/API_Metrology_Practical`
[14] http://ganglia.sourceforge.net/
[15] http://opentsdb.net

collected from computer systems. Written on top of HBase[16], the Hadoop[17] big data storage, OpenTSDB allows to collect thousands of metrics from thousands of hosts, applications and services, at a high rate (every few seconds). It provides a fine grained, real-time monitoring of the machines and manages the storage and analysis of collected data through a time series approach. The great benefit of using time series approach, along with the availability of temporal data, is that OpenTSDB proposes predefined functions for visualizing data in graphs and gives statistics on the data collected.

Enabling the collection of machine traces along with workload traces would have two benefits. First it would be a source of information on node failures. Then, it would give an information on what is the load of the machine apart from the load induced by the jobs. A job, even running alone on a single node is disturbed by external factors like the system load. In most of the cases, the system load should not impact too much the jobs but in case of a node problem, this system noise can start to perturb the job. A simple example would be a local disk that starts to have intermittent failures. With a performance decreased or worse, a jitter in the read and write speeds, the impact on the jobs can be huge if they require I/O operations. The correlation of data from node load and jobs load would help detecting potential problems.

Failure Traces: The Failure Trace Archive

The Failure Trace Archive (FTA)[74][18] is a public repository of availability traces of distributed systems that aims to facilitate the design, validation, and comparison of fault-tolerant models and algorithms. This work proposes a standard format for failure traces, more than 20 failure traces from different distributed systems, differing in scale, volatility and usage, along with scripts and tools for their analysis. Data collected take into account many aspects of system and hardware failures. Amongst information provided are start time and end time of the failure, system and node affected, as well as categorized information on the root cause of the failure. This can be software, hardware, network, I/O, infrastructure or even human. One of the goals of this archive is the integration of other types of data or models like workloads, for the study of the impact of platform failures.

The complete description of the format is available on the FTA format description page[19]. This work and its datasets have been used as a basis in many works on failure analysis. The complete list is available on the FTA publications page[20] (more than 20 publications based on these datasets). Amongst these works, [95] used a trace from Los Alamos[21] computing center to analyze root causes and rates of failures in

[16] http://hbase.org/
[17] http://hadoop.apache.org/
[18] `http://fta.scem.uws.edu.au`
[19] `http://fta.scem.uws.edu.au/index.php?n=Main.FTAFormat`
[20] `http://fta.scem.uws.edu.au/index.php?n=Main.Publications`
[21] `http://www.lanl.gov`

an HPC center composed of several clusters. An other work using a FTA dataset [65] analyzed the availability of resources in grid infrastructures and the performance degradation due to these failures. This analysis was based in data collected from the Grid'5000[22] infrastructure.

As PWA is an archive for workload traces, FTA is an archive for failure traces. The importance of such archives, only by the number of publications that follow, is undeniable. The collection of logs stored is an important base of knowledge. The problem is that logs from the PWA and FTA are not from the same platforms or not at the same periods. In the current state, it is not possible to correlate these traces and drive a deep study on the impact of failures on the workload. Either providing a trace from the same platform and period in both archives or a trace that already combines the two sources of information would enable such studies. This justifies the integration of (at least) node failures in a workload trace.

I/O Traces

Several repositories of I/O traces from storage, applications or nodes are available. They aim at helping the creation of model of loads for storage machines or model of I/O access for applications. Amongst the existing repositories are:

- Advancing storage & information technology[23]: it is an effort to create a worldwide repository for storage-related I/O traces.
- Maryland Applications for Measurements and Benchmarking of I/O on Parallel Computers : seven parallel I/O intensive applications were traced, capturing all message passing activity and context switches. This data is described and available in [86]
- Parallel Log Structured File System PLFS[24]: these are traces form different applications, recorded by the underlying file system (PLFS). They include both I/O and checkpoint information.
- Trace Data to Support and Enable Computer Science Research[25]: traces were collected by executing MPI_IO_TEST to provide I/O based traces.

I/O operations is something that is not easy to integrate in a workload trace. We have different kind of I/O. First there are the I/O that jobs do on the local disks of the nodes they have been allocated by the RJMS. We generally call this "local I/O". Then, we have the I/O that jobs do on the nodes but on a partition of the disk that is mounted from a remote DFS. These I/O operations have few local impact (some caching may be used) but have a remote impact on the DFS, whether it is centralized or not. Moreover, associated to I/O data are metadata. These metadata

[22]Grid'5000: `http://www.grid5000.fr`
[23]IOTTA Repository. `http://iotta.snia.org/repository/FAQsContributeTraces/`
[24]Parallel log structured file system. `http://institutes.lanl.gov/plfs/maps/`
[25]Los Alamos National Laboratory. `http://institutes.lanl.gov/data/tdata/`

do not account for in the same way as regular data. In decentralized distributed file systems such as Lustre[26], metadata is stored on a single node whereas data is distributed across the different nodes that compose it. In such systems, the impact of data and metadata is not the same on the DFS [9]. An I/O pattern with large metadata access will disturb the DFS more than large data access [98].

The study of such I/O repositories and works on I/O on DFS enables us to have a more precise view of what information should be integrated in a workload trace. As a first observation we can tell that local I/O should be differentiated from distant I/O in the workload trace and that metadata is also a potential problem as its impact is different from the regular data load.

At a higher level, other types of data such as the type of I/O (collective or file-per-process) or checkpointing information might also be an interesting knowledge to collect. This may lead to multi-level traces that would describe all the complexity of the I/O and communications and not only the raw data read and write sizes.

1.4 GCT, SWF, GWF, Getting the Best of Each Approach

SWF and GWF

SWF and GWF are relatively close from each others. Since GWF is based on SWF and extends it for grid workloads, it is not surprising. However, GWF not only adds the grid level, it also provides several interesting modifications. First the idea of a database friendly file format that provides logs in both flavors is nice. It enables an easier data management with the use of the power of database tools. SWF was already database friendly in its organization, but GWF exploits this feature to provide the data in database tables. Second, it adds information on network and local disk usages and requests. This is also a good thing. Third it redefines how resources consumptions are expressed with a list of comma separated pair *"Description of the Resource":"Consumption"*. Although this approach is very convenient as it requires only one field to describe all the resources consumptions, it suffers from two drawbacks. The first one is the fact that network and local disk I/O consumptions are already provided in other fields. These consumptions are the averages per processor. It would have been more coherent to use the key pair to describe these consumptions. Here there is potentially a duplication of information. The other problem with this approach is that it is not normalized as it does not respect the Database 1st Normal Form [21] (1NF, see Section 4.1 for definition). 1NF constraint states that values of attributes should be atomic. A list of comma separated key pairs is not considered as being atomic. As GWF is not really a database format it is however acceptable to relax the constraint. GWF requires that the "Description of the Resource" should be

[26] `http://www.whamcloud.com/lustre/`

described more explicitly in the GWF trace itself or in the GWA. This ensures that data in the trace should be understood by the other trace users.

In an extension, GWF also provides the notion of advance reservations. An advance reservation is the possibility for a user to submit a job that will be scheduled at a given date, potentially far from the submission time. This information is important to add to the workload trace format as the study of the time spent in the queue by the jobs should not take into consideration such jobs. Their wait is not due to a wait for resources availability but for the date to launch the reservation on the platform.

To sum up, GWF adds several interesting aspects to SWF like the possibility to add more types of consumptions for the resources and the notion of advance reservations which is a crucial information for better understanding jobs wait times in the workload analysis.

GCT

Despite the fact that both SWF (and thus GWF) and GCT are workload traces formats they have different approaches.

SWF and GWF are, more or less, usable in a database. However this should be taken cautiously as we have seen that GWF does not respect 1NF. GCT as for it, has a relational database trace format, composed of several tables that are linked together with unique identifiers. GCT proposes to embed in the trace format, information about the nodes (failures and configurations) and the job consumptions. The modular approach of database enables the format to be richer and enables to add many information without overcharging a main central file as it would be if we add GCT information to SWF. This approach is more extensible and maintainable. Providing consumptions of the resources in a temporal way is also very interesting and will allow jobs characterization. However, GCT still lacks some important information such as submission queue and job walltime.

As for SWF, GCT also misses several important information like the notion of advance reservations (provided by GWF), energy consumptions and DFS usage consumptions by the jobs.

Chapter 2

Tools for Workload Analysis and Workload Traces Provided

Thanks to the PWA, many workload logs for different types of clusters with different configurations and usages are now available. In order to ease the process of analyzing HPC cluster workloads we developed a set of tools to share with the community. This originated the Evalys-Tools project provided in a git repository that collects all these efforts. This repository is available at:
git clone https://forge.imag.fr/anonscm/git/evalys-tools/evalys-tools.git

This repository is organized as follows:

- Xionee tools collection.
- Workload Logs Collection provided in the SWF format.
- Recipes for the Kameleon[34] reproducibility tool.
- Scripts and README files to automatically deploy and experiment on the Grid'5000 platform.

2.1 Xionee

Xionee[1] is a collection of tools that is composed of three parts. It covers several usages from the generation of SWF workload logs from different RJMS to the statistical analysis of workloads and the experimental replay of workload traces.

In Xionee, for the workload analysis part, we provide R functions and scripts that will work with any workload log that respect the SWF format. With these, the analyst will be able to load in a dataframe on a R session an SWF workload log. Once loaded, several R function will provide the compute of the system's utilization, queue size and jobs arrivals along time. Graphical functions will provide a visualization

[1]word derived from Greek word χιόνι that means *Snow*.

of this information in graphs. Several other functions are available to compute the Cumulated Distribution Function (CDF) of Wait Times and Slowdown, to convert to/from timestamps and Human readable dates, and functions dedicated to the extraction of users information.

The idea of these R functions is to provide primitives for the analysts to ease the analysis process. The main file one should look at first is the *"analysis.R"* file, with the *"graph_usage(swf_file)"* function that loads an SWF file and plots in a single figure several graphs: system utilization, queue size, jobs arrivals, starts and ends and wait times CDF.

An other part of Xionee is the conversion to SWF from different RJMS logs and traces. Currently RJMS supported are: Slurm (from command line or from accounting database) and OAR (from database). An other tool dedicated to extract and convert to SWF the available information from the *"pjstat"* command for the K Supercomputer[2] at RIKEN[3]. This tool is provided in the second part of the Evalystools project with workload logs. Converters from RJMS logs to SWF are written in Ruby for about 1500 lines of code.

The third part of Xionee is *"riplay"*, a tool dedicated to the experimental replay of workload traces in their original context. This tool is described and used in Part II on Chapter 3.

2.2 Workload Logs Collection

Along with Xionee collection of tools, we provide several production machine workload logs that we collected and converted to the SWF format. These workload logs have been obfuscated from private data and can be integrated in the PWA for redistribution. Workload logs provided are:

- Curie.
- Stampede.
- K Supercomputer.
- 4 OAR[14][4] RJMS based mid-sized clusters from France and Luxembourg.

Curie

The Curie supercomputer[5], operated by the French Atomic Energy Research Center CEA[6], is the first French Tier-0 system open to scientists through the French

[2]http://www.fujitsu.com/global/about/tech/k/
[3]http://www.aics.riken.jp/en
[4]OAR: http://oar.imag.fr
[5]http://www-hpc.cea.fr/en/complexe/tgcc-curie.htm
[6]http://www.cea.fr/english-portal

participation into the PRACE[7] research infrastructure.

Curie is offering 3 different fractions of x86-64 computing resources for addressing a wide range of scientific challenges and offering an aggregate peak performance of 2 PetaFlops. It was ranked 11th in the November 2012 top500 list[8], then 15th in the June 2013 list.

Curie is managed by Slurm RJMS and since its upgrade in February 2012 it is composed of 3 partitions:

- Fat Nodes: 360 S6010 bullx nodes, each housing 4 eight-core Intel Nehalem-EX X7560 @ 2.26 GHz CPUs, 128 GB of memory; for a total of 11 520 cores.
- Thin Nodes: 280 bullx B chassis housing 5,040 bullx B510 nodes, each with two 8-core Intel Sandy Bridge EP processors (E5-2680) 2.7 GHz, 64 GB of memory; for a total of 80640 cores.
- Hybrid Nodes: these nodes combine Intel processors and Nvidia GPUs. This partition is composed of 16 bullx B chassis, each with 9 hybrid B505 blades. Each blade has 2 Intel Westmere 2.66 GHz processors and 2 Nvidia M2090 T20A GPUs, for a total of 288 Intel + 288 Nvidia processors. The Intel processors have 4 cores each. The Nvidia GPUs have 512 cores and 6 GB of on-board memory. Nodes in this partition are allocated in "exclusive mode" i.e. a job request for one or several full node (2 processors and 2 GPUs).

Thus in its last configuration Curie has a total of 93,312 CPU cores and 147,456 GPU cores. Among the cpu cores, only 92,160 can be independently allocated as the Hybrid partition is in exclusive mode.

The Curie trace has been recently released and is available for use in the PWA: `http://www.cs.huji.ac.il/labs/parallel/workload/l_cea_curie/index.html`. This trace is composed of 773,138 jobs, submitted between February 2011 and October 2012. However, in the first year only one partition was available (Fat Nodes) and the full capacity of the system was attained only after February 2012.

Curie's owner: CEA, is particularly interested into the analysis of Curie workload that result from this trace construction. Thus, Curie workload characteristics have been more carefully studied and are presented in the following Figures and Tables.

Figure 2.1 presents the utilization of Curie platform between February 2011 and October 2012. The upgrade of the platform is easily observable in February 2012. Before the upgrade, the number of available cores was 11,520 against 93,312 after the upgrade. In this figure, it is very noticeable the fact that the platform is highly used most of the time, with short and almost regular downtime periods. It is also particularly remarkable the fact that the maximum number of cores from the accumulation of the 3 partitions is never hit. The platform has never been fully allocated in its whole. This might be the cause of nodes failures, however we cannot verify this, as failures data is not available. The average utilization of Curie after its

[7] http://www.prace-project.eu/
[8] http://www.top500.org/site/50414

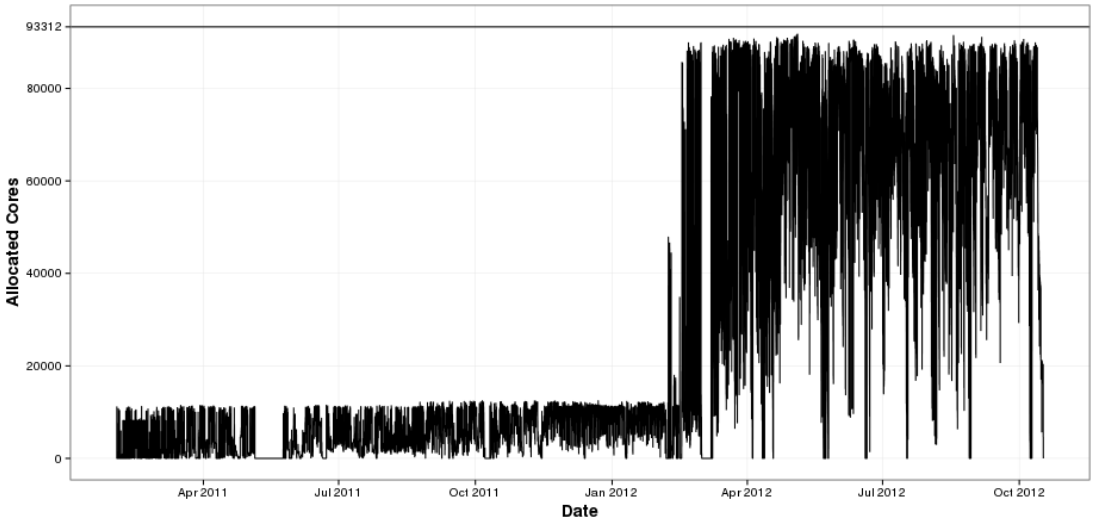


Figure 2.1: Utilization for the Curie Trace.

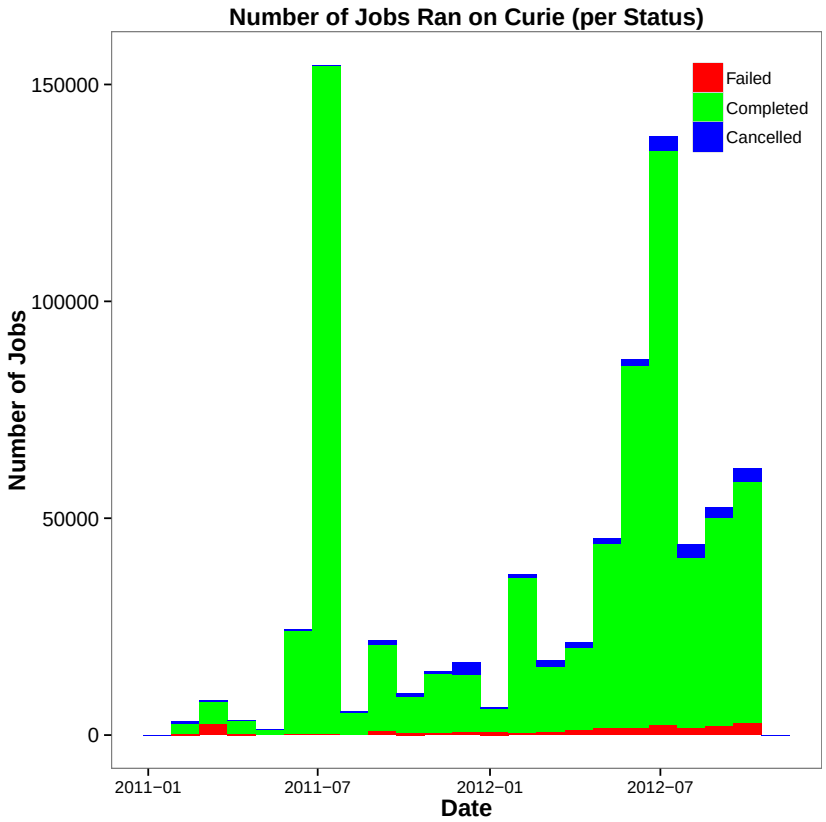
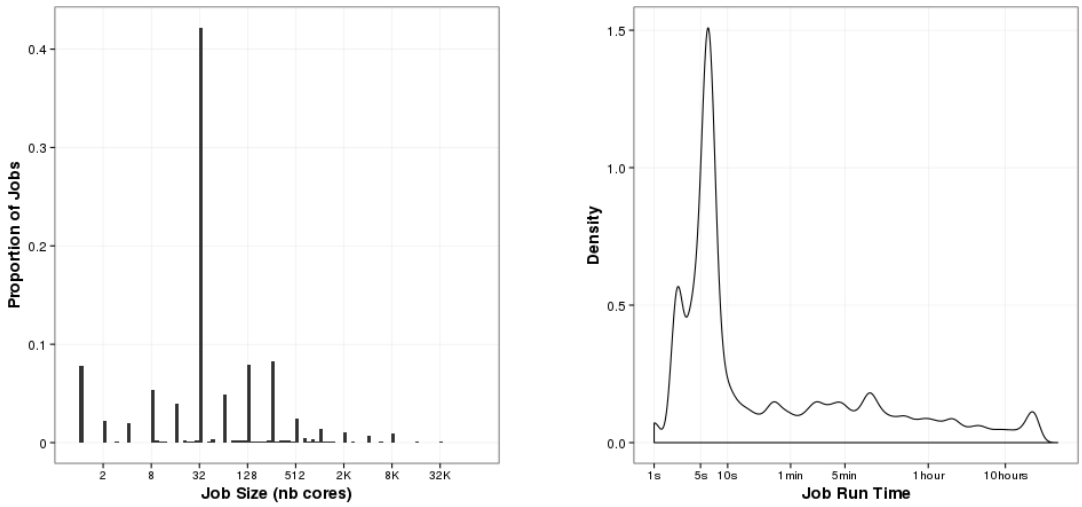


Figure 2.2: Curie Monthly Jobs regarding their Status.

upgrade (February 2012) is 75.7%, which is quite high regarding the utilization of other platforms that are available in the PWA.

In Figure 2.2 is presented the monthly number of jobs regarding their termination status. In July 2011, a high peak in the number of jobs is mainly caused by a single user. The case happens again in July 2012 but for an other user. However despite a large number of jobs submitted by these two users, the impact in terms of number of allocated cores is quite low. We also observe that in Curie, very few jobs are canceled or end in error regarding the volume of jobs that completed successfully.



(a) Distribution of Job Core Allocations for Curie Trace.

(b) Density Function of Job Run Times for Curie Trace.

Figure 2.3: Distribution of Job Core Allocations and Job Lengths for Curie Trace.

Figure 2.3 presents the distribution of core allocations (Fig.2.3a) and density function of run times (Fig.2.3b) of the jobs. The most probable number of cores a job requests is 32, which is also the first quartile and the median. Other common values are 1, 128 (3rd quartile) and 256. Maximum value is 79808 cores allocated to a single job. The information of the high probability of having a job that requests for 32 cores (2 nodes) is very interesting as it can be used as a scheduling optimization of resources pre-provisioning by Slurm. This information, combined with the probability of the inter-arrival times should enhance the scheduling efficiency and provide a faster resource attribution to the jobs. Prediction techniques used in this context might be an interesting path to follow for workload management.

In Figure 2.3b, it is observable densities over 1 (up to 1.5). This is not a mistake, unlike a probability, a density function, or also Probability Density Function (PDF) can take on values greater than one. It is actually the integral of the region (i.e. the probability) that is between 0 and 1. Density means probability per unit value of the

random variable. More generally, to get the probability of a random event within some range, it is necessary to integrate the PDF over this range.

The density of jobs run time is highly centered around 5 seconds and the values above are more sparse. When looking at data more carefully we observe that 68% of the jobs run for less than one minute and 81% less than 10 minutes. Almost 90% of the jobs run for less than an hour. Curie's jobs have very short run times and this should be studied deeper to understand the root cause of this phenomenon as we have seen that few jobs ended badly.

Inter-arrival Deciles									
10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
0s	0s	1s	3s	4s	5s	8s	28s	80s	434 hours

Table 2.1: Curie Inter-arrival Times Deciles

Slowdown Deciles									
10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
1	1	1	1	1	1.004	1.5	3	8.7	3.4e+05

Table 2.2: Curie Slowdown Deciles

Tables 2.1 and 2.2 present the deciles for jobs inter-arrival times and slowdowns respectively. We observe that in Curie jobs tend to arrive rapidly within the minute after their predecessor submission. High inter-arrival times are due to maintenance periods. We also observe that many jobs have a very good slowdown. 70% of them wait in the queue for half their run time. This is a very good performance result for Slurm. We also observe however, very high slowdowns. This is not necessarily caused by a bad system performance. As in Curie advance reservations[9] are enabled, this can simply be due to this feature. This observation highlights the fact that advance reservations should be treated separately from the rest of the workload and the information of whether a job is one of these have to be included in the workload trace.

To summarize, Curie has a high utilization percentage, with few jobs failures. Curie's jobs are small and short and are submitted close from each others. The slowdown is very good for 70% of them. The job turnover in the platform is thus quite fast.

[9] An advance reservation is the submission of a job at the present time to be run at a given date. Thus, the time spent in the queue by the job is not due to the wait for available resources but to the wait for the date to launch the job on the platform.

Stampede

Stampede[10] is a cluster from the Texas Advanced Computing Center (TACC). It is ranked 6th more powerful machine in the June 2013 Top500 list[11]. The TACC Stampede system is a 10 PetaFLOPS (PFLOPS) Dell Linux Cluster based on 6,400 Dell PowerEdge server nodes, each outfitted with 2 Intel Xeon E5 (Sandy Bridge) processors and an Intel Xeon Phi SE10P Coprocessor (MIC Architecture). They come with 32GB of "host" memory with an additional 8GB of memory on the Xeon Phi coprocessor card. There are an additional 16 large-memory nodes with 32 cores/node and 1TB of memory for data-intense applications requiring disk caching to memory and large-memory methods.

The aggregate peak performance of the Xeon E5 processors is 2 PFLOPS, while the Xeon Phi processors deliver an additional aggregate peak performance of 7 PFLOPS. The system also includes large-memory nodes, graphics nodes (for both remote visualization and computation), and dual-coprocessor nodes. Stampede has a total of 102,400 processing cores and 205TB of memory, managed by Slurm RJMS.

The Stampede trace is a 3 month extract, from January 2013 to the end of March 2013 for a total of 457,540 jobs. A longer trace should follow and will be available in the PWA as soon as possible.

Figure 2.4 shows the utilization of the platform for this period. We observe very regular downtimes probably due to periodic maintenances. We also observe that the average utilization (57%) is much lower than for Curie. This can be explained by the fact that Curie is an older platform, where users are used to its specificities and can benefit for the full power of the platform.

As for Curie, we observe that the maximal capacity (in terms of cores allocations) of the cluster is never reached.

Figure 2.5a shows that in Stampede, jobs request mainly for a power of 2 number of cores with a high probability of 16 cores. Other core allocations are more rares and the frequency of the number of jobs decreases when jobs ask for more cores.

Figure 2.5b shows that run times tend to be centered around a little less than 10 minutes, however it is also observable a large dispersion of the values. A quite important number of jobs tend to run for about a full day long.

Inter-arrival Deciles									
10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
1s	1s	2s	3s	3s	4s	8s	18s	40s	94 hours

Table 2.3: Stampede Inter-arrival Times Deciles

When looking at inter-arrival times and slowdown deciles in Tables 2.3 and 2.4, we observe two things: jobs submissions tend to be very close from each other but

[10] `http://www.tacc.utexas.edu/resources/hpc/stampede`
[11] http://top500.org/system/177931

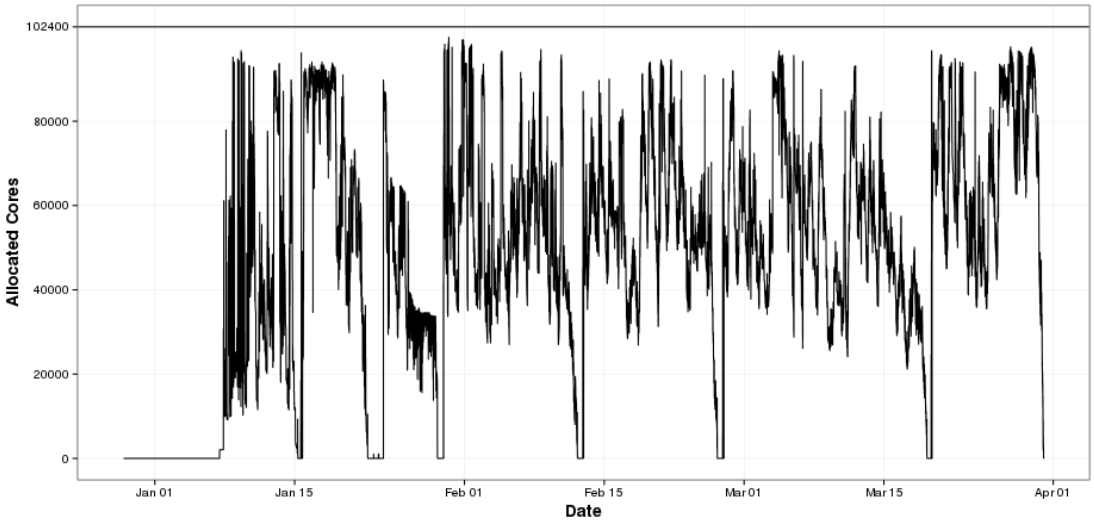
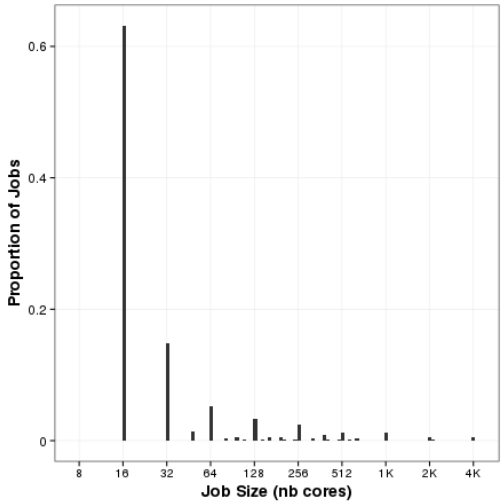
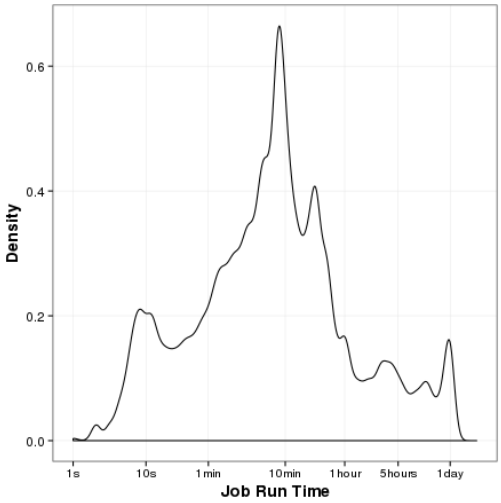


Figure 2.4: Utilization for the Stampede Trace.



(a) Distribution of Job Core Allocations for Stampede Trace.



(b) Density Function of Job Run Times for Stampede Trace.

Figure 2.5: Distribution of Job Core Allocations and Job Lengths for Stampede Trace.

this does not impact the system too much as the slowdown is 1 for 80% of the jobs. This means that jobs almost never wait in comparison to their run times. This phenomenon is probably due to the low average utilization of the cluster that enables job to be scheduled very quickly as few other jobs are waiting in the queue.

Slowdown Deciles									
10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
1	1	1	1	1	1	1	1	1.07	1.33e+05

Table 2.4: Stampede Slowdown Deciles

To summarize, Stampede has a lower utilization than Curie, this impacts strongly the slowdown which is 1 for most of the jobs. Stampede jobs tend to be small, requesting for a number of cores as a power of 2 but their durations are more varied than for Curie and tend to be longer. They arrive in the system at a rate on the same order than in Curie.

K Supercomputer

K Supercomputer[12] is a very large supercomputer from Riken[13] in Japan. It was ranked 1st in the Top500 list[14] two times in a row in June and November 2011 (after its upgrade). It is ranked 4th in the June 2013 list. The K computer achieved recently the goal of a LINPACK score of more than 10 PFLOPS.

K is composed of more than 80,000 8-core SPARC64 VIIIfx 2.0 GHz processors contained in 864 cabinets, for a total of over 640,000 cores. In addition, 5184 I/O dedicated nodes bring K to a total of 705,024 cores. K is managed by a RJMS specially developed by Fujitsu for this platform.

The K trace is a 1 month and a half extract, with 41089 jobs between December 2012 and end of January 2013. Figure 2.6 presents K utilization for the 1.5 month trace. We observe that K has also a lower utilization percentage than Curie and Stampede. Its average utilization for the trace extract is 45.4% if we consider the whole machine and 50% if we consider only the 80,000 computing nodes and not the I/O nodes.

Figure 2.7 presents the distribution of cores allocations and density function of run times of the jobs. We observe that jobs in K request for a larger number of cores than in Curie and Stampede. This comes from the fact that in K, all the nodes allocations are in exclusive mode: a job requests for a full node or several full nodes, no allocations of jobs in the same node are possible. The most probable values (ordered by decreasing probability) are 8 cores (1 node), then 768 cores (96 nodes), then 512 (64 nodes) and 96 cores (12 nodes).

Concerning the run times of the jobs, we observe that in K, jobs tend to run much longer than in Curie and a little longer than in Stampede. There is also a high density of jobs running for several hours. However, the dispersion of data is even

[12] `http://www.fujitsu.com/global/about/tech/k/`
[13] `http://www.nsc.riken.jp/index-eng.html`
[14] `http://www.top500.org/system/177232`

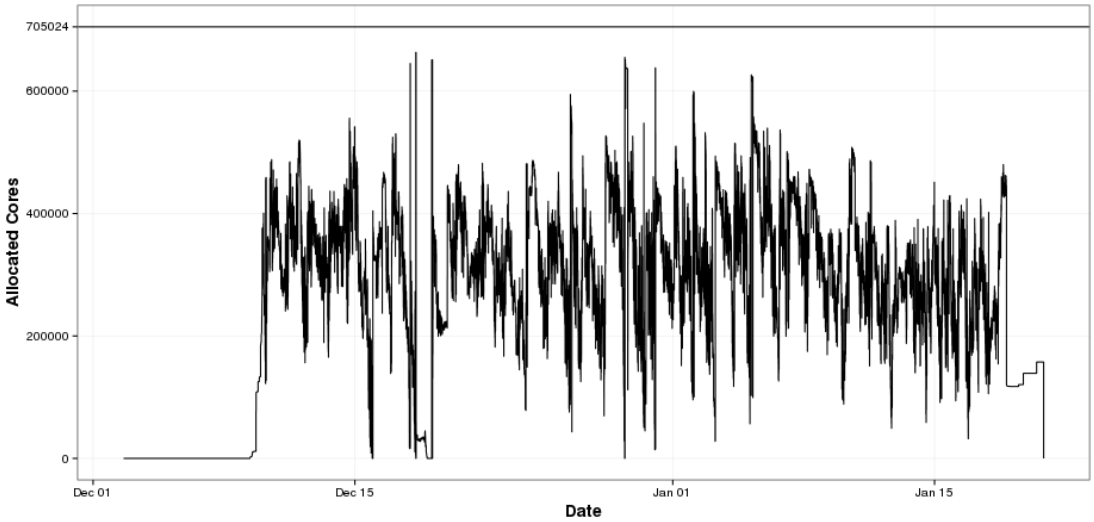
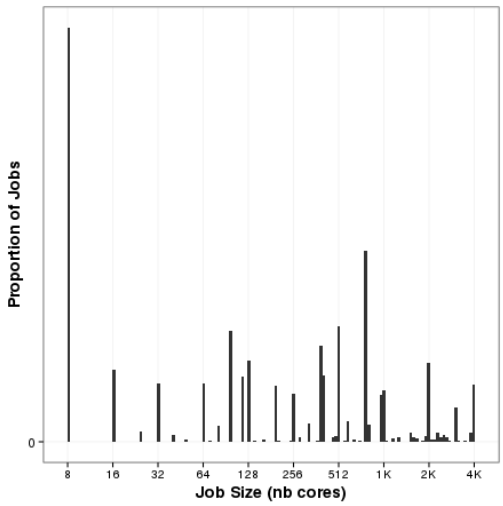
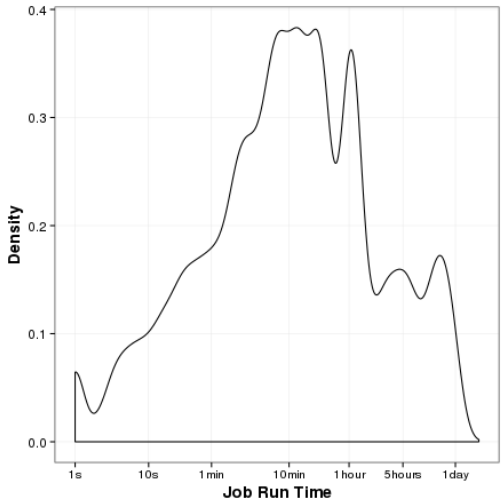


Figure 2.6: Utilization for the K Supercomputer Trace.



(a) Distribution of Job Core Allocations for K Trace.



(b) Density Function of Job Run Times for K Trace.

Figure 2.7: Distribution of Job Core Allocations and Job Lengths for K Supercomputer Trace.

more important than for Stampede. This dispersion is principally located for higher run time values.

Inter-arrival time deciles are presented in Table 2.5. We observe that in K, the throughput of jobs arrivals is lower than for the 2 other clusters. Median is 27 seconds against 4 seconds for Curie and 3 seconds for Stampede. However, 60% of the jobs

Inter-arrival Deciles									
10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
2s	4s	7s	14s	27s	46s	1min 13s	2min	3min 41s	81 hours

Table 2.5: K Supercomputer Inter-arrival Times Deciles

Slowdown Deciles									
10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
1.08	1.22	1.52	2.1	3.1	4.8	8	17.8	54	1.57e+05

Table 2.6: K Supercomputer Slowdown Deciles

still arrive within the minute of their predecessor's arrival.

Table 2.6 shows slowdown deciles. We observe that compared to Curie and Stampede, slowdown values are higher in K. Despite the fact that jobs run longer than in the 2 other platforms, the system utilization is also lower and such high slowdowns are strange. In Stampede, the slowdown was very close to 1 in most of the cases because the utilization was low, few jobs were waiting in the queue and thus jobs' wait times were low. As K's utilization is lower than Stampede's, we should observe the same behavior. We could think that the problem can come from bad user estimates that cause backfilling problems [6, 112, 80], thus lowering the global utilization and impacting negatively jobs' wait times. However when comparing the difference of these estimates and the actual run time, we discover that they are not that bad estimates: 75% of the jobs have an overestimation of less than 2 hours. They even are better than in Curie where only 25% of the jobs have an over estimation of only 2 hours. Unfortunately, user estimates are not available for Stampede to continue the comparison.

The origin of such high slowdowns remains unknown although one probable track is the RJMS itself. Using a workload trace from K and replaying it with Slurm as done in the methodology proposal in Part II might help discovering if the root cause is the scheduling, the resource management or an other factor.

Other Traces

Along with the earlier presented workload traces from large clusters we provide 4 workload traces from mid-sized production computing cluster. Two of them: Foehn and Gofree belong to the CIMENT platform[15] located in France and are presented in Chapter 3.4. The other two: Chaos[16] and Gaia[17] belong to the University of Luxembourg. These four clusters are all smaller size clusters and have about few

[15]CIMENT Project. https://ciment.ujf-grenoble.fr/
[16]`https://hpc.uni.lu/tiki-index.php?page=Chaos`
[17]`https://hpc.uni.lu/tiki-index.php?page=Gaia`

hundred cores. They are managed by the OAR [14][18] RJMS. Their analysis will become available in the evalys-tools repository by the end of August 2013, and is not presented here for the moment. The analysis along with the comparison with the other traces observed will be integrated in the final version of this document in September 2013.

[18]OAR: http://oar.imag.fr

Chapter 3

Jobs Resource Consumption

3.1 General Context

Studying the RJMS workload has become a widely used method for HPC systems valuation. The workload in the RJMS context can be defined as the set of all individual jobs that are processed by the system during a specific period of time. With workload traces, one can reconstruct the scheduling, determine the system's computing resources utilization, compare different systems and their workloads according to several metrics like *Average Weighted Wait Time* and *Average Weighted Response Time* as used in [37]. This study can lead to the construction of models as proposed in [82] and more generally described in [41].

However, looking only at the workload traces may not be sufficient. To get a better understanding of the use of such systems we need to look at both how the jobs interact with the RJMS but also how they consume the allocated resources. In other words, we need to look at both the workload and the jobs activity on the resources.

The idea of associating RJMS workload traces to the jobs resource consumptions has been mentioned in several works. [94] points out the fact that the System Utilization metric, commonly used in system evaluation misses information about the computer sub-systems (network, memory, processor) usage. According to [121] it is also necessary to take into account other characteristics such as I/O activity which have a big impact on the global performance of HPC systems. However, the aforementioned association has never been fully achieved.

The System Utilization metric is the ratio of the computing resources allocated to the jobs over the available resources in the system. In fact, this metric corresponds more to the *System Resource Allocation* as it does not give information about the physical utilization of the resources. In this paper we will focus on the Resource Utilization metric which reflects the ratio of the consumption of a resource by a job over the amount of resource allocated by the system to this job. For each resource type: core, memory, IO, we will look at their utilization by the jobs on the clusters.

The following Section presents several works related to Workload and Jobs tracing. The first part presents briefly D. Feitelson et al. works on workload traces then several methods for retrieving information about the jobs consumptions are presented. Then

in Section 3.3 we present the solution chosen to collect jobs resource consumptions data. Section 3.4 describes the clusters on which data was collected and the trace characteristics. Section 3.5 presents the analysis of the results of the different resources consumption metrics. Finally, Section 3.6 discusses the results and gives some perspectives.

3.2 State of the Art

Jobs Consumption Traces

Two options are possible to produce a trace of the jobs consumptions. First, giving for each job, for each type of resource (memory, processor, disk, network), a single representative value of the resource consumption. The question is thus *"How representative of the real consumption this value is?"*. Or, for each job, giving a trace of the resource consumptions over time. This can be viewed as a monitoring of the jobs consumptions.

In [68], R. Jain proposes one possible classification of the different monitoring techniques. In this classification, the monitoring process can be *event-driven* or by *sampling*. The event-driven method is very efficient and there is no monitoring overhead if the event is rare. The sampling method as for it, is well adapted for frequent events but a loss of data captured is inevitable and a frequency resolution has to be chosen. This section presents the state of the art of the existing monitoring systems that can be adapted to our context.

Monitoring provided by the existing batch schedulers.

Several RJMS provide an embedded system for monitoring the jobs consumptions.

SLURM [119][1]**:** uses different mechanisms to know which processes are members of a SLURM job, then monitors their consumptions. One of them is the cgroup isolation mechanism that requires a kernel version above 2.6.24, the other one is based on the Linux process table.

OAR [14][2]**:** provides a mechanism that monitors the jobs consumptions in terms of processor, memory and network. As for SLURM, it uses different mechanisms to know which processes are members of an OAR job, these are the cpuset or cgroup features of the kernel and require a recent kernel version. This data collection is not automatic and is triggered at user request.

LoadLeveler[3]**:** allows the user to gather information about resource consumption. It offers different ways to consult this information and create different class of reports. As for OAR, this mechanism is not automatic.

All these approaches are linked to a particular batch scheduler system. For all of them, the traces generated give for each job a single value (generally the mean)

[1]SLURM. https://computing.llnl.gov/linux/slurm/
[2]OAR: http://oar.imag.fr
[3] LoadLeveler: http://www-03.ibm.com/systems/software/loadleveler

for each type of resource consumption. This is problematic for several reasons. First for the memory consumption, the mean does not give valuable information. At least the maximum value and how many times this value was reached must be reported. Then for the IO consumption, the variance of the reads and writes can vary a lot and the mean is still not representative enough. We prefer to adopt a monitoring of the resources over time, that will give us all the details of the evolution of the consumptions and thus enable a deeper analysis of the cluster's resources utilization.

Monitoring and Profiling Tools.

Many monitoring and profiling/tracing tools exist in the literature, but our approach needs specific requirements. First, we need to collect the jobs resource consumptions, this is a completely different process than machine monitoring. Then, we need to collect these data in a temporal way. This means that data has to be collected over time along with the jobs executions. Last, we want to collect jobs consumption data on production clusters which implies two constraints: the impact of the monitoring on the compute nodes has to be negligible; and the setup of the monitoring on the cluster has to be as simple as possible. On a running production cluster it is not acceptable to deeply modify the configuration of the nodes, nor install or update too many softwares; and the update or modification of the nodes' kernel is not possible.

Means and maximum values are not sufficient for analyzing the behavior of the applications over time, we need a more fine-grain view of their consumptions. Several systems like Ganglia[84] and Nagios[63] have been developed to monitor a system or a cluster infrastructure but are resource centric, they perform their monitoring at the machine level. Our approach needs to be job centric to enable us to extract the resource consumptions per job as proposed in [22]. Other approaches more application-oriented exist as [87] which gathers an application consumptions in an online manner taking advantage of two tools, TAU[100] as the data collector and Supermon[106] as the transport layer. Or [99] which monitors an application at runtime, with a low overhead, allowing it to study the overhead penalties incurred by Linux Systems in the execution of HPC applications. The difference between our approach and the systems mentioned above is the fact that we aim to monitor all the jobs executed in the cluster, generating a trace of the resource consumptions over time. This requires a very lightweight tool, capable of monitoring all the jobs with a low overhead and with an amount of data generated that can be easily stored and processed. These conditions were not respected by these tools.
Linux Kernel tools like Performance Counters[4] or CGroups[5] would have been a good approach. However this is not a possible option in our case as this would have implied the update of the compute nodes kernel, which was not acceptable regarding to the production policy.

[4]https://perf.wiki.kernel.org
[5]http://www.kernel.org/doc/Documentation/cgroups/cgroups.txt

We also tested several event based tools like IPM[52], TAU[100], ltrace[6], but either their overhead was too high or the amount of data collected was too important (hundreds of MB for a single application run) to enable a global workload analysis. An event-driven approach would thus be too intrusive and the amount of data produced would have been too big.

The approach chosen was then a sampling monitoring, enabling us to restrict the amount of data collected and to set the precision required.
Because of the production constraints we chose to collect information about the consumptions from the */proc* subsystem (as used in the OpenTSDB[7] approach, see Section 1.3). In this virtual file system (present on any Linux system), for each process is given its consumption of resources such as memory, processor, or IO on the Distributed File System (DFS). With this method, no modification nor software installation is needed on the compute nodes.

3.3 Resource Consumption Capture

This section presents the approach used in order to instrument the system, which allowed us to have more detailed information about jobs resources consumption during their execution. This monitoring process is divided into two parts:

- The monitoring of the jobs' processes execution in each compute node of the cluster. This is performed by a monitor daemon running on each node.
- The collection of all the traces generated by each job.

Monitor daemon

It is important to understand that we do not want to depend on the synchronization of measures. A centralized clock that forces measure times would be too intrusive for the system. Instead, every node is responsible for measuring each job every minute and the clocks between the nodes are synced (this is a common requirement in HPC). Every measure is tagged with its timestamp.

The monitor daemon, written in Perl, gathers resource consumption information for a job that is running on the machine. This information includes every process involved in the job and their resource consumption values obtained from */proc* directory. We first validated this method by making several tests with jobs whose behavior were well-known to ensure that information given by */proc* was consistent with what we expected. Collected values and trace format are described in Tables 3.1 and 3.2. Unfortunately, energy consumptions were not recorded in the measures as this information is not available in the platforms used for the information collection. In order to know which processes belong to the job, the monitor looks into the

[6] A library call tracer. http://linux.die.net/man/1/ltrace
[7] http://opentsdb.net

cpuset directory, which is the interface to the *cpuset*[8] kernel mechanism to provide job isolation. Therefore with all this information collected, the monitor generates a trace file per job. The cpuset feature is supported by most of the RJMS as PBS, Loadleveler, Torque[9], Slurm[10], OAR[11], it is the most portable method for collecting the jobs processes.

We wanted the finest granularity possible without generating a big perturbation on the nodes and a minute step allowed us to have a perturbation under 1% in the worst case. It is thus a good compromise information/intrusiveness.

As these traces will be mixed with RJMS traces, a 60 seconds granularity seems reasonable as the scheduling decisions and the resources allocations are in the order of the minute.

To be as lightweight as possible, the monitoring tool does not reformat or process data during the capture. Values taken from */proc* are written directly to the log for the overhead to be in the analysis process, not during traces capture.

Intrusiveness and Sampling Evaluation

This section presents the evaluation of our approach in terms of disruption in the system. The cost of retrieving information from the RJMS and converting it to SWF format is null as it is done post mortem. We will thus study the cost to monitor the jobs. It is very important for the validity of the data collected that the monitoring process does not interfere with the jobs themselves. It is obviously also important for the validity of the jobs results. We chose that the job overhead should be strictly less than one percent. All the tests were performed over a machine Intel Xeon E5420, with 8 cores and 24 GB of main memory. We evaluated the implementation under a 100% processor load of the machine, with one process per core and measured the monitor overhead by using the Sysbench [75] multi-threaded benchmark tool. The evaluation ran 8 processes, each one checking a prime list up to 18000. Results are presented in Table 3.3.

We also evaluated the noise for more than 8 processes with a 60 seconds frequency to have an idea of the scalability of the solution. The results are shown in Table 3.4. The disruption caused by the monitoring was found to be almost linear regarding the number of processes to monitor in these conditions. Given that there were four times more processes than cores in the 32 processes benchmark, the result of a 0.35% overhead is acceptable.

The memory used by the daemon is very small (few MB) and it does not use the network during the capture. The reads and writes are a few KB every minute on the local disk and the Distributed File System is not solicited so the IO perturbation is negligible.

[8] http://www.kernel.org/doc/man-pages/online/pages/man7/cpuset.7.html

[9] http://www.clusterresources.com/torquedocs21/3.5linuxcpusets.shtml

[10] https://computing.llnl.gov/linux/slurm/

[11] http://oar.imag.fr/sources/2.5/docs/documentation/OAR-DOCUMENTATION-ADMIN/#cpuset-feature

Trace Fields	
Name	Description
Time	Unix Time Stamp in seconds
JOB ID	Job id assigned by the batch scheduler
PID	PID of process that belongs to the job
Node ID	Provenance of the capture
Measure	Measure as presented in Table 3.2

Table 3.1: Trace Format

Values Captured (extract)	
Name	Description
command	Name of the binary executed
vmPeak	Peak virtual memory size (KB)
vmSize	Total program size (KB)
vmRss	Resident Memory (KB)
vmSwap	Size of swap usage (KB)
syscr	Number of read syscalls
syscw	Number of write syscalls
read_bytes	Bytes read from the DFS
write_bytes	Bytes written to the DFS
core	Core utilization percentage
receive	Bytes received from the network
transmit	Bytes transmitted to the network

Table 3.2: Data collected during a measure

Collecting mechanism

Data is gathered using dedicated jobs that collect the logs when the system is underloaded. This operation does not need to be frequent; week-ends, holidays, night-times or maintenance periods can be used for this. We used a special maintenance period during holidays for this purpose. Each job requests a whole node and is in charge of gathering and compressing the trace files generated so far by the jobs executed on that machine. Then it sends them to a dedicated node in charge of storing the trace data.

Off-line data processing

Once we have collected data about jobs resources consumption we need to bind it to the RJMS log to be able to compute the resource utilization ratio. This ratio is at a given time, for a given job, the amount of resource consumed by the job over the amount of resource allocated by the system to this job. To make it simpler to use and redistributable, we first build the SWF file from the RJMS log. The SWF is an

Sampling Freq.	Mean	Min	Max	Overhead
No	573	573	573	0%
60 sec	573.6	573	574	0.10 %
30 sec	574.4	574	575	0.24 %
10 sec	577	576	578	0.69 %

Table 3.3: Overhead (in terms of perturbation in the application execution time) vs. sampling period. All values are in seconds

16 processes				
Type	Mean	Min	Max	Overhead
No monitoring	513	513	513	0 %
monitoring	514.1	514	515	0.21 %
32 processes				
Type	Mean	Min	Max	Overhead
No monitoring	512	512	512	0%
monitoring	513.8	513	516	0.35 %

Table 3.4: Overhead (in terms of perturbation in the application execution time) vs. number of processes monitored per job. All values are in seconds

abstract view of the log, this format gives us the amount of resource allocated by the system for each job.

As measures are not guaranteed to be synchronized between nodes during the capture, a shift in the measures dates can appear. Thus we first re-sample the measures then bind them to the RJMS logs with R[12]. The whole re-sampling and binding process is available in a R script available in the evalys-tools repository, see beginning of Chapter 2 for more information.

3.4 Experiment Environment

Data was collected from production clusters of the Ciment project[13]. This project aims at gathering several computational infrastructures to provide computing power to users in different disciplines like environment, chemistry, astrophysics, biology, health, physics. The CiGri project relies upon the OAR RJMS to submit the jobs on the clusters. All the Ciment clusters are managed by the CiGri[14] lightweight grid software that gathers clusters resources to make them available as a grid for the users.

The monitoring tool was implanted and run on two of the Ciment production clusters: *Foehn* and *Gofree*. As they come from the same grid, their respective

[12]http://www.r-project.org/
[13]CIMENT Project. https://ciment.ujf-grenoble.fr/
[14]CiGri Project: http://cigri.imag.fr/

workload should not be too different from each other. Their characteristics are detailed in table 3.5. A short summary of data collected is presented in Table 3.6.

	Foehn	**Gofree**
Brand	SGI	Dell
CPU Model	X5550	L5640
Nodes	16	28
CPU/node	2	2
Cores/cpu	4	6
Memory/node	48 GB	72 GB
Total storage	7 TB	30 TB
Network	IB DDR	IB QDR
Total Gflop/s	1367.04	3177.6
Buy date	2010-03-01	2011-01-01

Table 3.5: Clusters characteristics

	Foehn	**Gofree**
Capture start	2011-06-01	2011-05-24
Capture months	8	7
Log Size	2.6 GB	3.1 GB
Number of jobs	53662	20093
Besteffort jobs	50403	15596
Normal jobs	3259	4497
Active users	54	35

Table 3.6: Trace summary for both clusters.

One particularity of the Ciment platform is that users can submit a particular type of job: the "besteffort" jobs. These jobs are scheduled in a dedicated queue. They have a very low priority and can be preempted whenever a "normal" job arrives and needs the resource. The goal of such jobs is to maximize the cluster utilization as they are plentiful. Generally, these jobs request one core to the RJMS. These special jobs are multiparametric sequential jobs. A user using these kind of jobs has generally several instances of the same application running in different jobs with different parameters.

3.5 Analysis

In this section we analyze the consumption of the jobs from data collected during the capture regarding their requests. This analysis is done for the following metrics: core utilization, memory utilization and IO activity on the network file system. The two clusters although belonging to the same platform are located in two different laboratories. When submitting a job, the users of the platform can both choose on

which cluster the job will run or do not specify anything. In this case it is the local cluster which is selected by default. As we know, most of the users do not request specifically a cluster or tend to prefer the local cluster. Thus we analyze separately the results from the two clusters. Moreover, jobs from the **normal class** and **besteffort class** have very different patterns. Besteffort jobs are multiparametric jobs that generally request only 1 core (although we know that some besteffort jobs request several cores). They are supposed to be processor intensive with few memory usage and few IO activity. Normal jobs are generally parallel jobs requesting several nodes/cores. Their consumption patterns are not really known and are probably more varied. Hence, the following analysis of the jobs consumptions is done on the two classes separately on each cluster.

To address the analysis of the consumptions in a global way for each class we look at the resources utilization distribution. This will give us an idea of the different existing patterns.

Squashed Area Ratio.

Along with the distribution of the utilizations we will also consider the impact of a particular phenomenon into the global system activity. We present thus the Squashed Area metric (SA) that represents the jobs execution activity. [105] defines SA as the total system's resource consumptions of a set of jobs, computed by:
$SA = \sum_{j \in jobs} allocated_cores_j \times run_time_j$.

Thus, for a job or a set of jobs, we look at the SA ratio of this set regarding the workload SA of its class (i.e. the proportion of this set area over the total class area). This enables us to determine if a particular phenomenon is significant enough regarding the rest of the workload.

Core.

In data collected we have for each job j, for each core c allocated to j, the proportion of core consumed by the processes running on c and belonging to j along its execution. We denote this value: $core_consumption_j^c(t_i)$, with t_i being the date of the measure i. The value reported by $core_consumption_j^c(t_i)$ is the average core consumption for core c by the job j between t_{i-1} and t_i.

Thus, we can compute for each measure date t_i the total amount of core consumed by a job with:
$core_consumption_j(t_i) = \sum_{c \in allocated_cores_j} core_consumption_j^c(t_i)$.
Then, with n being the number of consumption measures taken for j, we can compute per job its core utilization mean by:

$$core_utilization_mean_j = \frac{\sum_{i \in [1,n]} core_consumption_j(t_i)}{allocated_cores_j \times n}.$$

In the following analysis we consider the distribution of these values to identify different utilization patterns.

Memory.

The OAR version managing Foehn and Gofree clusters does not provide a memory isolation of the jobs on the nodes. This feature in OAR is only available with the Cgroup feature of the kernel enabled which is not the case. Thus it is interesting to look whether jobs use a lot of memory or not. Indeed, a job which uses intensively the memory could disturb other jobs sharing the same node.

We introduce the *theoretical_memory_per_core* value which is equal to a node memory over the number of cores on the node. As each cluster has an homogeneous architecture, this value is the same for all the nodes in a cluster. Foehn and Gofree does not have the same number of cores and memory per node but their *theoretical_memory_per_core* is the same and is equal to 6GB. We consider thus that a job which consumes less than this value per allocated core is not disturbing for the other jobs.

Then, for a given job we know its theoretical maximum memory. This value is equal to $theoretical_memory_per_core \times allocated_cores_j$. Thus we know that jobs that consume more than 100% of this value are potentially disturbing other jobs by using too much memory on at least one node.

For the analysis of the memory utilization we will look at two sub-metrics:

- The mean memory used by a job (computed by the same method than the core utilization mean) vs. its theoretical maximum memory. This tells us how the job behaves on average regarding its theoretical maximum memory.
- The maximum memory used on the cores allocated to this job. This will tell us if the job used more than the *theoretical_memory_per_core* on one of its allocated cores or not.

File System IO.

As the Distributed File System (DFS) is a central component of a cluster we are also interested into its usage patterns. On Gofree no user has complained about slow IO on the DFS. This is not the case for Foehn where users have reported several IO problems. Unfortunately we could only monitor the IO utilization on the Distributed File System on the Gofree cluster and this for only two months and a half. Gofree DFS consists in an NFS server that serves the users' home directories through the GigaBit Ethernet network. We tested its capabilities in terms of bandwidth with the IOR[15] benchmark[97]. We used IOR in a single file/single client mode with file sizes of 64MB and blocks of 4KB. 4KB is the default block size. 64MB and 64KB are the most frequent write sizes in Gofree (see Figure 3.15). We chose 64MB instead of 64KB because this last value is too small (only 16 physical writes) to have a correct

[15] http://sourceforge.net/projects/ior-sio/

idea of the bandwidth. We repeated the test 10000 times and got a max speed of 103MB/s for the writes and on the order of 1000MB/s for the reads. The mean write speed was 98MB/s with a standard deviation of 5. Several other tests with higher values of file sizes gave us similar results. For concurrent sequential tests IOR showed that the bandwidth was more or less fairly divided between the different writing processes. For parallel tests (with MPIIO API, one file per process, up to 48 processes) IOR showed a maximum value close to 96MB/s which is a little less than with the single process mode but still in the same bandwidth range than the mean speed of sequential writes. This gives us an approximative idea of the global bandwidth of Gofree DFS.

The read bandwidth is quite high (close to a local file system bandwidth) and generally the type of IO which are problematic are the writes, thus we will focus on the analysis of the writes patterns.

Foehn Cluster

Normal Class Jobs.

Figure 3.1 presents the distribution of the core utilization means for normal class jobs on Foehn cluster. We observe a peak in the distribution corresponding to a core utilization near 100%. An other peak, less important corresponds to a low core utilization (between 0% and 25%). However when looking at the SA ratio of these two peaks the tendency reverts. The SA ratio of the high utilization peak is only 12.7% of the total workload SA and the SA ratio of the low utilization peak is of 53.7%. For these jobs, very few of them consume a lot of memory (only 22 jobs). Among them only 5 consume more than their theoretical maximum memory. For these memory intensive jobs, the maximum memory used per core is 125% of the *theoretical_memory_per_core*. Figure 3.2 presents the distribution of the mean memory used per job. We can observe that memory intensive jobs are very rare on Foehn but exist.

Figures 3.3 and 3.4 present the maximum memory used on a core per job. We split the representation of the distribution in two groups for more clarity: up to 100% and more than 100%.

There are 31 jobs whose mean memory usage is over 100%. Unsurprisingly, jobs that have a mean memory usage over 100% are the same than the jobs that have a peak memory usage on an allocated core over 100%. These jobs have a peak value grouped around 140%. Only 1 job has a very high peak memory usage of 480%, it is a job that requested 2 cores and lasted for 18 minutes then was canceled by its user.

In Foehn, we observe that many jobs are using the cores computing power very efficiently and do not use a lot of memory. These jobs seem to be cpu bound. However their weight in the workload is counterbalanced by larger and shorter jobs (larger number of resources but with a runtime being a little shorter on average). Regarding the memory usage distribution, we can think that these jobs suffered from something else than memory problems. But when we look at the number of cores allocated

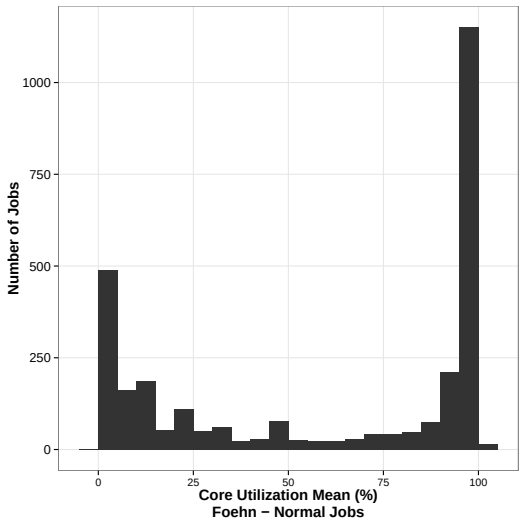


Figure 3.1: Distribution of the jobs' core utilization means.

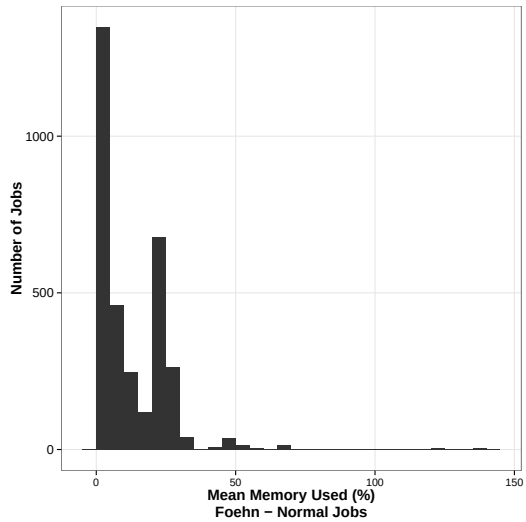


Figure 3.2: Distribution of the mean memory used by job.

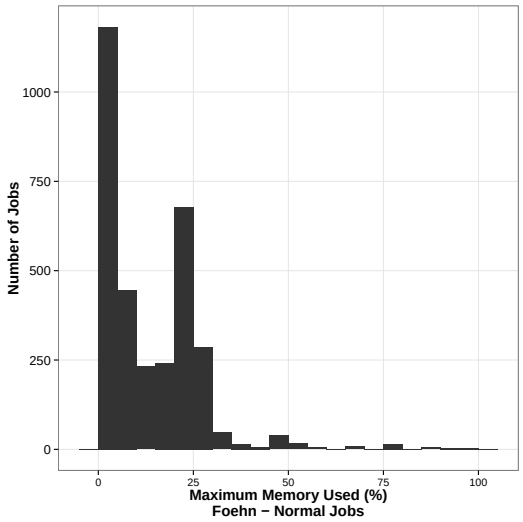


Figure 3.3: Distribution of the maximum memory used per job by an allocated core to this job, values⩽100%.

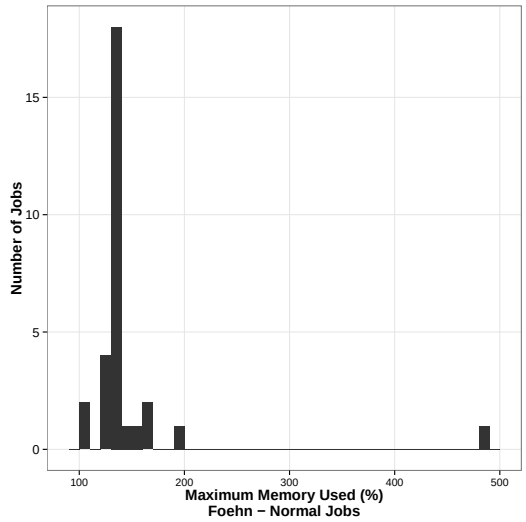


Figure 3.4: Distribution of the maximum memory used per job by an allocated core to this job, values>100%.

to the jobs of the two groups (jobs with average core utilization under 25% and jobs with average core utilization near 100%) we remark something very particular. The mean number of allocated cores is 4 times bigger for the jobs that have a core utilization in the lower peak. This can be an IO scalability issue with jobs waiting on IO, as we know that Foehn DFS has bandwidth problems, this having been reported several times by the users. However this can also come from an over-reservation of cores by the users. Sadly, we currently miss information to explain the reason of this

phenomenon.

Besteffort Class Jobs.

Figure 3.5 presents the distribution of core utilization for besteffort jobs on Foehn cluster. We observe two peaks, one corresponding to a core utilization mean around 25%, the other around 100%. For the lower utilization peak (from 0 to 25%), its SA ratio is 55.5%. The SA ratio for the jobs exactly in the 25% peak is 42.4%. For the jobs involved in the higher distribution peak, even though there are fewer, their SA ratio is about the same (40.8%). 99.2% of these jobs reserve only 1 core. These jobs never use more than the theoretical memory per core. 99.5% of them actually use less than 1/3 of this theoretical memory per core.

Only 6 users are involved in the low core utilization peak, but 5 of them are also involved in the high utilization peak. The user only present in the low utilization peak accounts for 41.9% of the besteffort SA and has a core utilization less than 25% in 99.4% of the cases. He reserves 4 cores in all his besteffort jobs. In almost all the cases his processes consume less than 310MB and never higher than 620MB.
After contacting the user, he explained that he noticed that if two of his jobs were running on the same node, the performances of the two jobs were very bad. After investigating the cause of this it was found that the application was memory intensive in terms of bandwidth. The maximum memory used was small compared to the amount of memory available on the node but when two jobs were accessing the memory at same time, there was a bottleneck in the access to the memory. The solution adopted by the user to avoid this performance loss was to reserve the whole socket (corresponding thus to 4 cores) even though the application only used 1 core. As Foehn nodes are NUMA, reserving the whole socket enabled the jobs to have their own memory slot and thereby not being disturbed. The problem here was not a misconfiguration of the job or a bad reservation request but a lack in the RJMS constraint description that forced the user to over-reserve.

Figure 3.6 shows that besteffort jobs do not consume a lot of memory on average. Only 6 jobs (not represented on the figure for clarity) have a mean memory utilization greater than their theoretical available memory. Their mean memory use is between 7 and 8 GB.

In Figures 3.7 and 3.8 we observe that besteffort jobs do not have big memory peaks. Only 6 jobs have peaks between 115 and 135%. These jobs are all 1 core jobs and five of them belong to the same user.

Except the singularity of the user reserving one entire socket and few jobs with memory usage peaks, besteffort jobs on Foehn tend to be cpu bound.

Gofree Cluster

Normal Class Jobs.

Figure 3.9 shows the distribution of the core utilization means for jobs in the normal class in Gofree. We observe a very particular distribution with four peaks

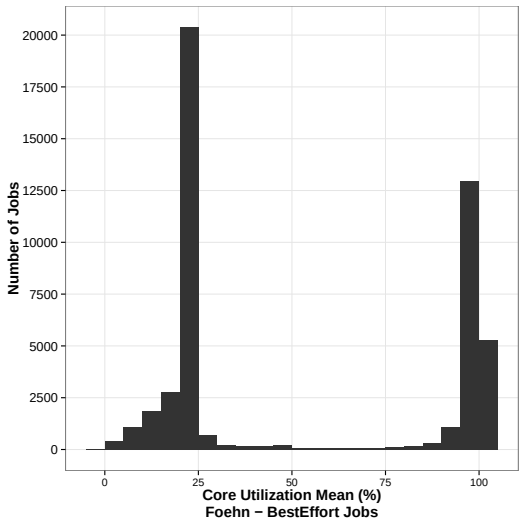


Figure 3.5: Distribution of the jobs' core utilization means.

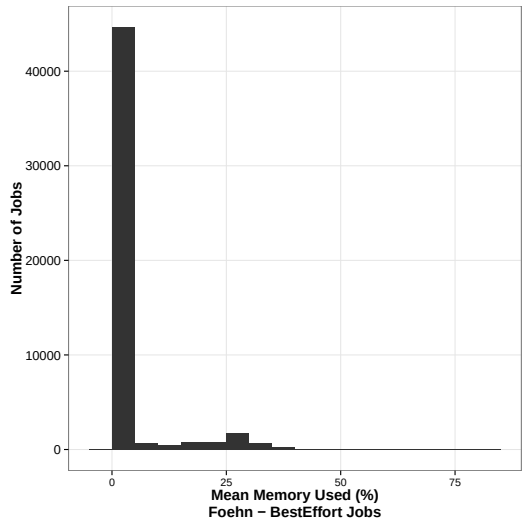


Figure 3.6: Distribution of the mean memory used by job.

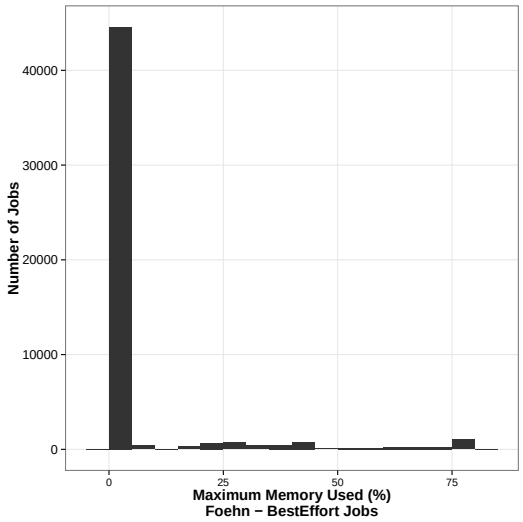


Figure 3.7: Distribution of the maximum memory used per job by an allocated core to this job, values⩽100%.

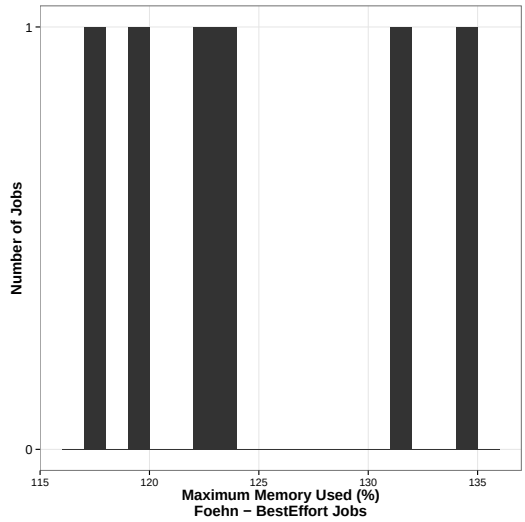


Figure 3.8: Distribution of the maximum memory used per job by an allocated core to this job, values>100%.

around 0, 1/4, 1/2 and 1 of core utilization mean. The most surprising is when we look at the number of cores allocated to these jobs vs their core utilization. The higher peak has a core allocation of 1 node in 50% of the cases (with a maximum of 1 node). The peak around 1/2 has a core allocation of 2 nodes in 85% of the cases (with a maximum of 2 nodes). The peak around 1/4 has a core allocation of 3 nodes in 48% of the cases and 4 nodes in 35% of the cases (with a maximum of 4 nodes). The jobs below 50% of core utilization are not an isolated phenomenon, 2/3 of the

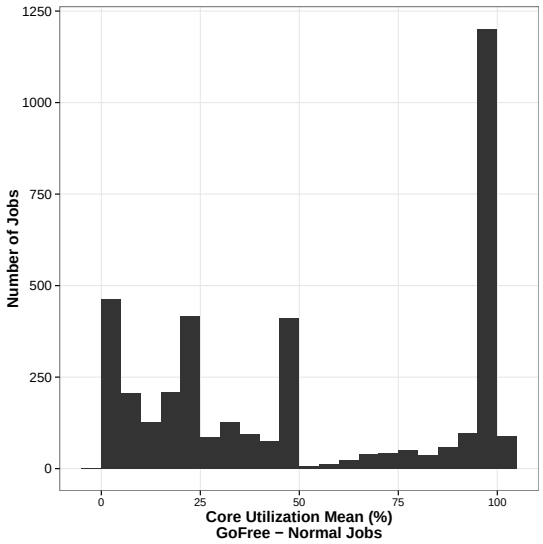


Figure 3.9: Distribution of the jobs' core utilization means.

users are involved in these jobs and their presence is distributed all along the trace. These users are also involved in the high utilization peak. The memory used for these jobs is low (less than 1/3 of theoretical memory) for 89% of them. Only 7 jobs use up to 100% of their theoretical available memory. However, despite the fact that these jobs have a low core consumption on average, a lot of them have both high and low core utilization along their execution. When we look at their core consumption over the time on all their allocated resources, we observe that actually half of them have also measures reporting a full global core consumption. This means that a lot of these low core consuming jobs also have temporary high core loads.

Figures 3.10 and 3.11 show us a memory utilization quite low for most of the jobs of this class.

This behavior of core utilization vs core allocation and the jitter between high and low core load is very noteworthy. This could come from jobs waiting for IO, however results of IO consumption on data collected during the IO capture period showed the DFS is usually not very stressed, see Section 3.5. Thus we suspect a scalability problem although we cannot say where is the bottleneck. However, as for Foehn, this can also come from an over-reservation of cores by the users. These two phenomena of low core utilization on both clusters will need further investigations and the jobs' knowledge of the users involved.

Besteffort Class Jobs.

Figure 3.12 shows that besteffort jobs in Gofree have a high core utilization. Jobs below 75% of core utilization represent 0.37% of the normal jobs SA and jobs below 90% account for 7% of the class' SA. Core allocation for besteffort jobs in 85% of the cases is 1 with other values being 4 and 6 (1 cpu socket). Figures 3.13 and 3.14 show

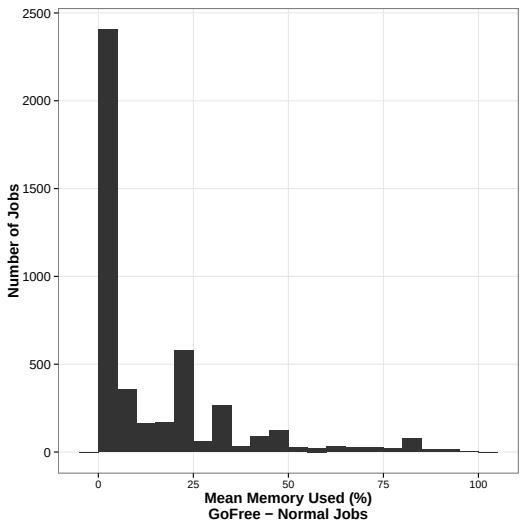


Figure 3.10: Distribution of the mean memory used by job.

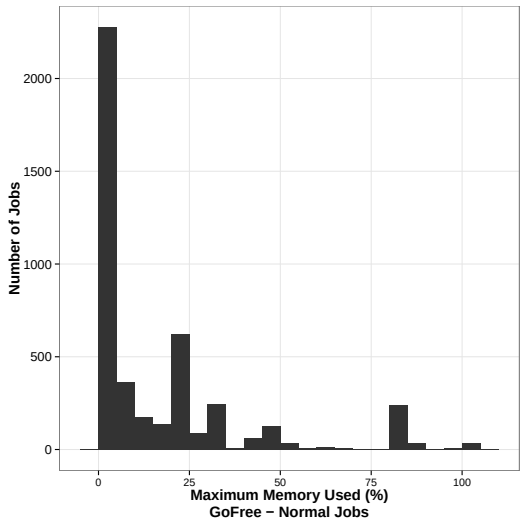


Figure 3.11: Distribution of the maximum memory used per job by an allocated core to this job.

us a low memory utilization on average and on maximum. Regarding these results we can say that besteffort jobs in Gofree are cpu bound.

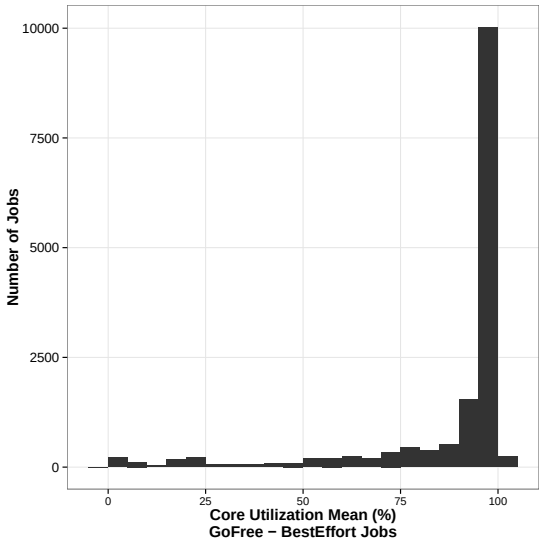


Figure 3.12: Distribution of the jobs' core utilization means.

Global IO Activity.

Figure 3.15 shows the distribution of the size of the files written by the jobs. We can observe a big peak at 64KB. This comes from many besteffort jobs that write

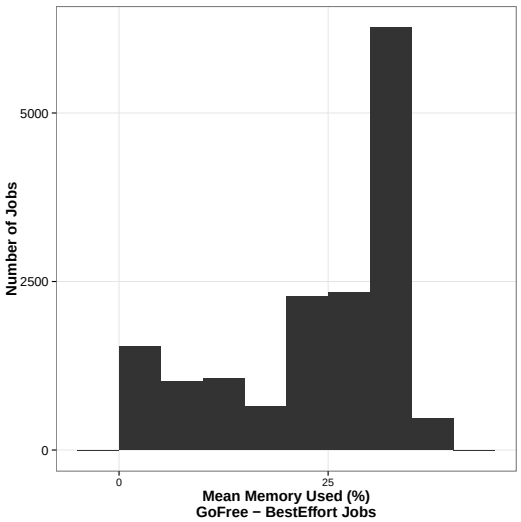


Figure 3.13: Distribution of the mean memory used by job.

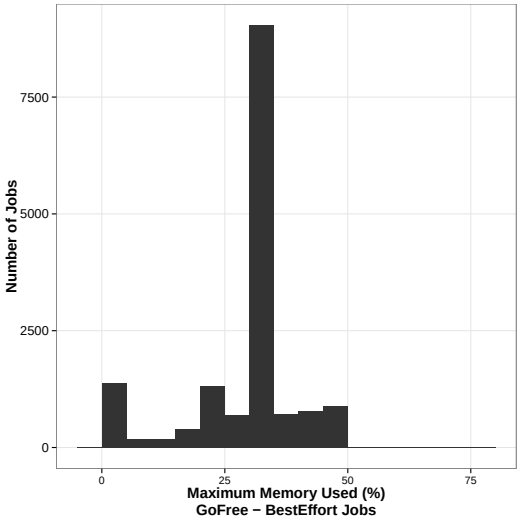


Figure 3.14: Distribution of the maximum memory used per job by an allocated core to this job.

small files. Generally, besteffort jobs tend to write either files of 64KB or 32MB. Normal jobs tend to write files of 4KB and 16MB.

Figure 3.16 presents the distribution of the aggregated writes by the jobs on the File System. We can observe that the most frequent sizes the DFS has to write in a minute is either small (between 4KB and 16KB) or medium (between 8MB and 64MB). It seems that the DFS is not really overloaded. Very few values are high (up to 6GB in a minute).

In order to see how the load is distributed along the time we plot Figures 3.17 and 3.18. Figure 3.17 presents the aggregated writes per day on the DFS. We can observe that the daily IO write activity is very irregular with intense periods and an empty period. The period with no IO activity occurred during holidays, there was almost no job submitted during this time.

Figure 3.18 plots the write load in terms of speed on the Gofree DFS server. In this figure, we isolate some remarkable values above the vertical line at 93MB/s. The value of 93MB/s has been chosen because it reflects the speed where the DFS might start to be overloaded. It corresponds to the mean write speed minus the standard deviation given by our bandwidth tests with IOR. We refer at the area above this line as the DFS hot zone.

We observe five ranges of dates where the DFS is in the hot zone. These dates are respectively the June 9, June 22, June 28, July 18 and August 8. The most remarkable period is July 18 where in a period of two hours the DFS reached the hot zone 10 times. When looking at the jobs involved in this activity we see that two jobs (belonging to two different users) where competing for IO on the DFS. The jobs where using 4 and 12 nodes. They didn't use much memory and their mean core utilization

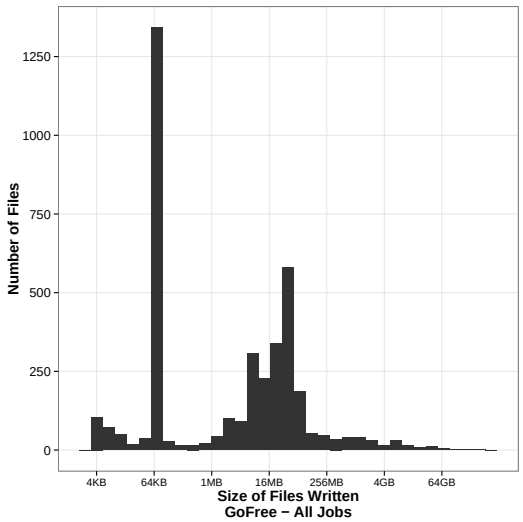


Figure 3.15: Size of Files Writes on the DFS on Gofree cluster for all jobs.

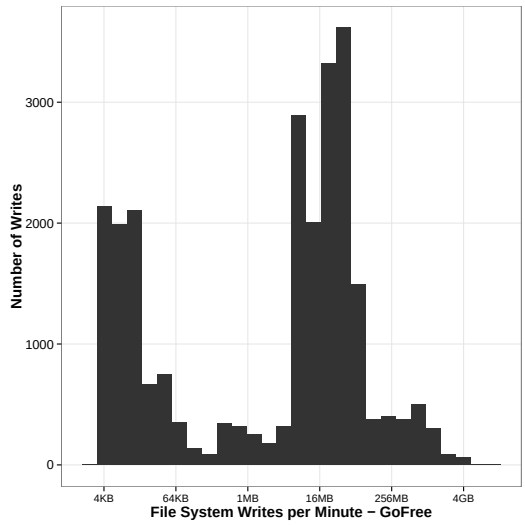


Figure 3.16: Distribution of the aggregated File System Writes sizes (per Minute) on Gofree cluster.

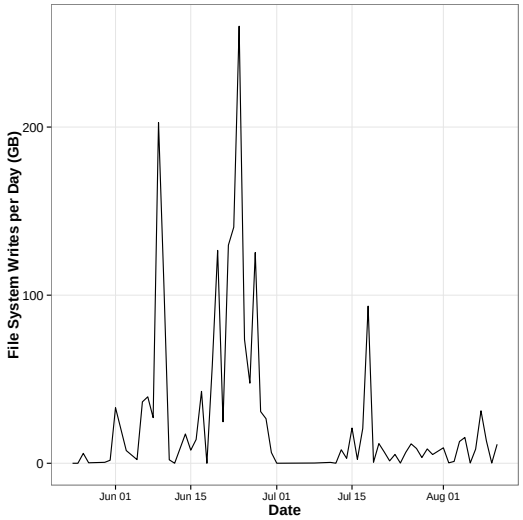


Figure 3.17: Daily aggregated writes on the DFS.

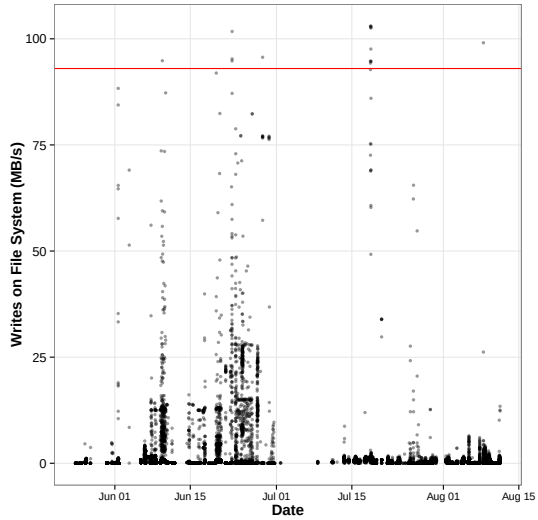


Figure 3.18: DFS load (write speed) on Gofree cluster.

was quite low. The job using 4 nodes was canceled by the user before its end. This particular scenario let's us think that this job that was canceled was suffering for slow IO and thus killed by its owner. The scenario is about the same during June 22 where four jobs where competing on IO. One ended, canceled by the user and one had an IO write pattern alternating big writes with periods of inactivity (few memory used, no core activity) probably being blocked on IO. However this event lasted for only 10 minutes. The other periods are less remarkable as the presence of the DFS in

the hot zone was brief (only one peak) with few jobs involved in IO activity.

3.6 Discussion and Perspectives

Monitoring the jobs resource consumptions and linking it to their resource allocation enabled us to detect some particular behaviors on the clusters. Moreover, the monitoring over time allowed us to have a more precise view of the resources utilization, particularly it enabled us to reconstruct the Distributed File System (DFS) write load along time. The study of the utilization for the different resource types gave us a better comprehension of our users' usage of the clusters. We are now able to propose some enhancements in our future scheduling strategies based on the observed usage of the system.

The problem of memory bandwidth risen in the analysis is very interesting because we can encounter the same lack of constraint description for memory, IO or network in most current RJMS. As the Squashed Area ratio of the user involved in this phenomenon weights a lot regarding the rest of the workload, the core computing power loss is important. Here it is not a resource quantity problem, but a problem concerning the bandwidth to access the resource. We cannot try strategies like scheduling other besteffort jobs on the same socket since we do not know their bandwidth usage patterns. A possible solution would be to define memory, IO and network bandwidths as resources in the RJMS in the same way cores are defined as resources. Thus we enable the user to reserve not only a set of cores but an amount of bandwidth on the node depending his/her need. However, this kind of guarantee is not possible nowadays.

It is also noteworthy that besteffort jobs on both clusters are core efficient with a low memory usage. This will be used as a scheduling tweak; besteffort jobs, whenever it is possible, should be scheduled on the same nodes. As we now are sure that they will not disturb with one another, we will pack these jobs on the same set of nodes to improve their efficiency. Besteffort jobs are killed whenever a normal job needs the resource they are running on. By packing them we reduce the fragmentation and thus the probability for them to be killed.

The problem of the DFS overload is also interesting. We observed that generally the DFS is not too loaded but when it happens and there are jobs competing for IO it ends with the cancellation of one of these IO intensive jobs. Giving the users the possibility to tag their jobs (e.g. as IO intensive) at the submission to the RJMS might prevent such situations, the RJMS will simply not schedule two IO intensive jobs at the same time.

The fact that a lot of normal jobs tend to have a low core consumption average, a low memory usage but also short high loads of core consumption, can be a symptom of jobs that require a high computing power for a limited period of time. The use of Dynamic Jobs [73] (jobs that change their resource needs along their execution) and the RJMS capability to treat these kind of jobs [72] would be an interesting feature to use in such a case.

Monitoring jobs resource consumptions revealed problems that were not visible with the sole System Utilization metric. Coupling jobs consumptions and RJMS logs enabled us to exhibit and quantify significant particular cases. On mid-sized computing centers, the study of the jobs resource utilization will become a necessary way to a deeper understanding of the users needs and their jobs consumption patterns. Particular patterns discovered by the analysis of the jobs resource utilization will lead to dedicated system setup and optimizations to improve both users and administrators satisfaction.

These works initiated the Colmet[16] software that takes over the idea of job centric monitoring. This tool aims at gathering several metrics of jobs consumptions on clusters, grids and clouds. Colmet uses the cgroup isolation mechanism to retrieve the processes that belong to a given job and uses the *taskset* kernel feature[17] to collect consumption metric samples. Colmet also uses HDF5[18] data model to store consumption data in a more efficient way and also propose machine monitoring techniques along with job monitoring.

[16] `https://gforge.inria.fr/projects/colmet/`
[17] `http://linuxcommand.org/man_pages/taskset1.html`
[18] `http://www.hdfgroup.org/HDF5/`

Chapter 4

Clusters Workload Trace Format Proposal

With more than 30 available logs from different machines managed by different RJMS, SWF led to several workload models being published and used in other works. With many references to the format itself (the work that introduced the format[16] was cited more than hundred times, several other publications followed and used this format as a basis), SWF has been the major standard for describing cluster workload logs. With the release of GCT format, radically different in its approach from former workload trace formats, it is now important to go back over some specifications of the SWF (and GWF) format.

GCT brought several major improvements as the information of machine status (configurations, failures and maintenance), jobs consumptions and resources constraints. However this format still misses some important points such as job requested time (also referred as walltime), the type of job (interactive, advance reservation or batch), inter-jobs dependencies and their think-times; these are information that are provided by SWF. GCT proposed an approach based on a database format (but implemented as a set of CSV files) composed of several tables, linked with unique identifiers. The benefit of this approach is the capability to easily merge different sources of information from jobs and machines and the facilitation of future evolution of the format.

In an extension, GWF provides the notion of advance reservations, which is an important information to collect in the trace. This format also enables to describe many types of consumptions for the resources.

Other types of resource consumptions that should be taken into account are energy and not only local I/O but also parallel and distributed. I/O in general have a strong impact on the jobs performance and on scheduling [92]. Scheduling strategies that take into account jobs highly demanding for parallel I/O tend to perform better [17].

In light of these differences, their respective advantages and our experience in the work of combining SWF traces with jobs consumption traces we can now propose a new cluster workload trace format that takes the benefits of the different approaches.

4.1 Which Constraints should we Adopt?

First, SWF is the basis of many works on workload analysis and its log collection is a huge base of knowledge. This is a work that we cannot loose. Moreover:

- The format is also simple to create and to use.
- Adding information on nodes failures in SWF is difficult. SWF is focused on jobs, not nodes.
- In many cases in production platform, information about jobs consumptions or machine failures or configuration is not easily available.
- Adding fields to SWF would break the standard and would make incompatible its different versions.

Changing SWF format by adding fields would either make the existing SWF obsolete or the new proposal unusable. Our proposition is to use SWF as a basis and to describe a relational database format, built on top of this basis. This will enable to embed missing information from other types of sources, as proposed by GCT, and to extend SWF without breaking the compatibility with existing works. The advantage of this proposition is, as for GCT, the possibility to be free in the way to implement the data structure, as long as it is based on the database format description. If a workload trace does not have all the information needed to be converted to the new format version it is not a problem as we use *job_id* for table joints, which is a mandatory information in SWF. Thus, in any case, the different sources of information can be associated. With the database approach, a partial implementation of the schema is possible.

In its workload archive, GWF provides data in two flavors: database dump and plain text. In our approach, we propose to use only the database version and to provide tools to convert from database to plain text (and in particular to SWF format) and vice versa. This enables to continue to work with data from the PWA. Using the database version enables to make data processing easier. Database engines are powerful tools and will allow to process data mining more easily than flat plain text formats.

For the format description we use a relational database format that strengthens the constraints a bit, but ensures data consistency, avoids redundancy and enables normalization.

Finally, this format proposal aims at being the first version of the extended format. We built it with the idea of future evolution, and the use of a database will allow to easily extend the format. Extensions will simply add tables or table fields.

Relational Database Normalization

To be consistent, the resulting relational database should respect the Third Normal Form (3NF) introduced by E.F. Codd [21]. A database is described as "normalized"

if it is in 3NF [23], this ensures that tables are free of insertion, update, and deletion anomalies. To verify 3NF, a database table has to respect the following constraints:

- each domain attribute should contains only atomic values. This means that a value should not be decomposed into smaller values (1NF) [32];
- every non-prime attribute of the table is dependent on the whole candidate key (2NF);
- every non-prime attribute is directly dependent on every superkey (set of attributes that compose the key).

Put simply, a common saying is that attributes should depend on:

- the key (1NF),
- the whole key (2NF),
- nothing but the key (3NF).

4.2 Format Proposal and Conversion

In this section we explain how to convert data from SWF files to a relational database format then, how to add information from other sources like consumptions and node failures to this format. The resulting tables all respect 3NF normalization.

SWF to Database

Converting SWF to a relational database is straightforward. The first step is the extraction of the header in a database table. The header cannot be included in the trace table itself as it does not contain information on the jobs but on the platform.

In Figure 4.1 we show the transition from SWF header plain text to database table. The first step is to add a field: *platform_id* that will be used in the other tables to identify a workload trace. As several workload traces can be stored in the database, we need this attribute to identify the relationship between tables. Thus, for most of the tables, this field will be a component of the key, coupled with *job_id* for the tables that concern the jobs information and with *node_id*, for tables that concern nodes information. The *node_id*, will identify a particular node in the cluster. We also take the occasion to remove the deprecated field "TimeZone" from SWF.

The second step is the externalization of the description of queues, partitions and notes to respect 3NF. They will give three separate tables. To later make the correlation between a given trace header and its queues, partitions and notes, we simply have to do a natural join on these tables on the field *platform_id*.

The third step, presented in Figure 4.2 is the conversion from the SWF trace to relational database format. As for the header, we add the field *platform_id* to be

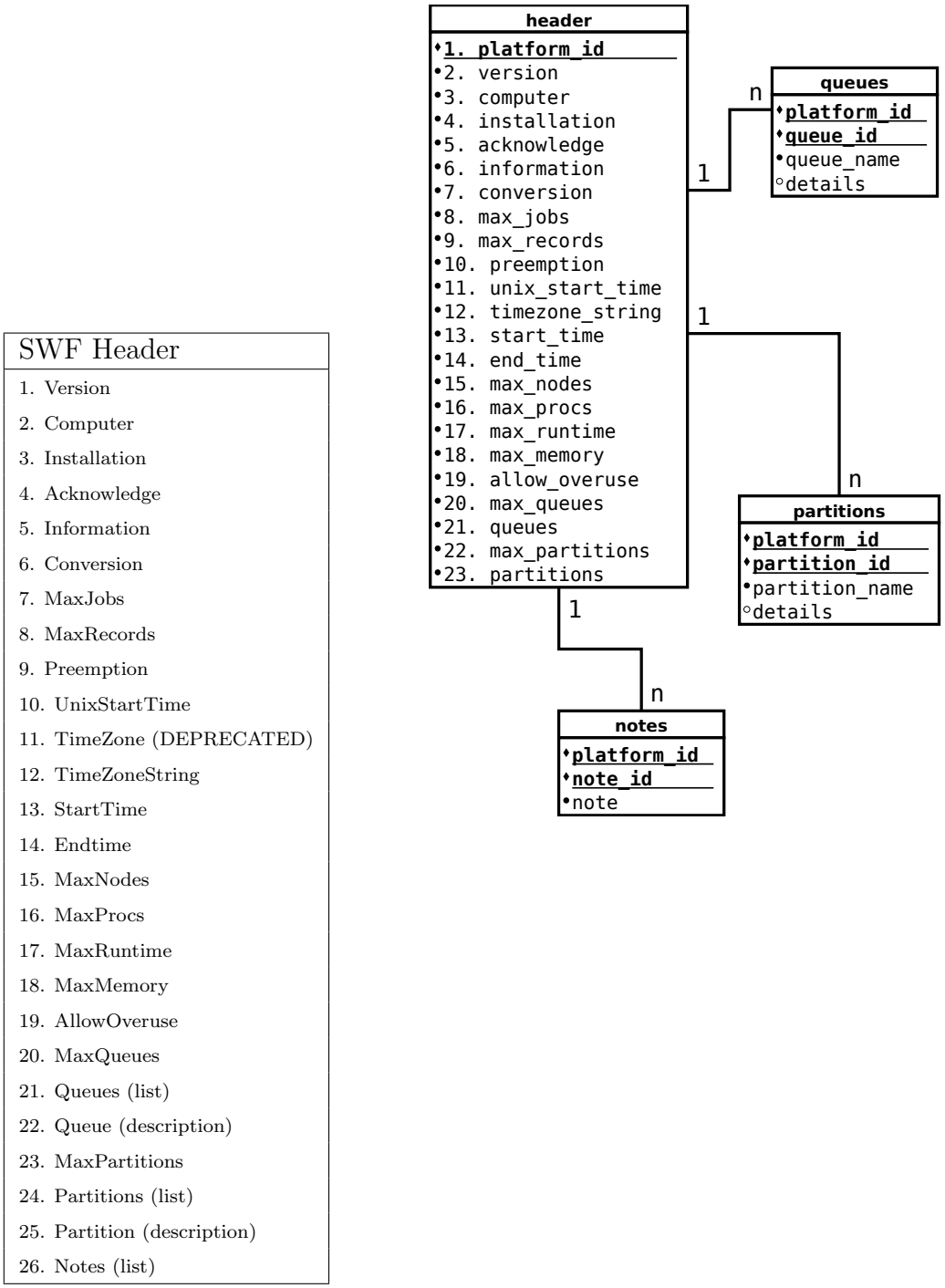


Figure 4.1: SWF Header and its Conversion to Database Format. Note the externalization of Queues, Partitions and Notes and the add of the field *platform_id* to be used as an identifier.

able to associate a trace with its header, and put a constraint on the *queue_id* and *partition_id* fields of the table. They should belong to one of the existing values for

this *platform_id*. The key will be the couple <*partition_id*, *job_id*> as several traces will probably have the same *job_id*. Thus, we will identify a job by this composed key that associates a job to a trace in the database.

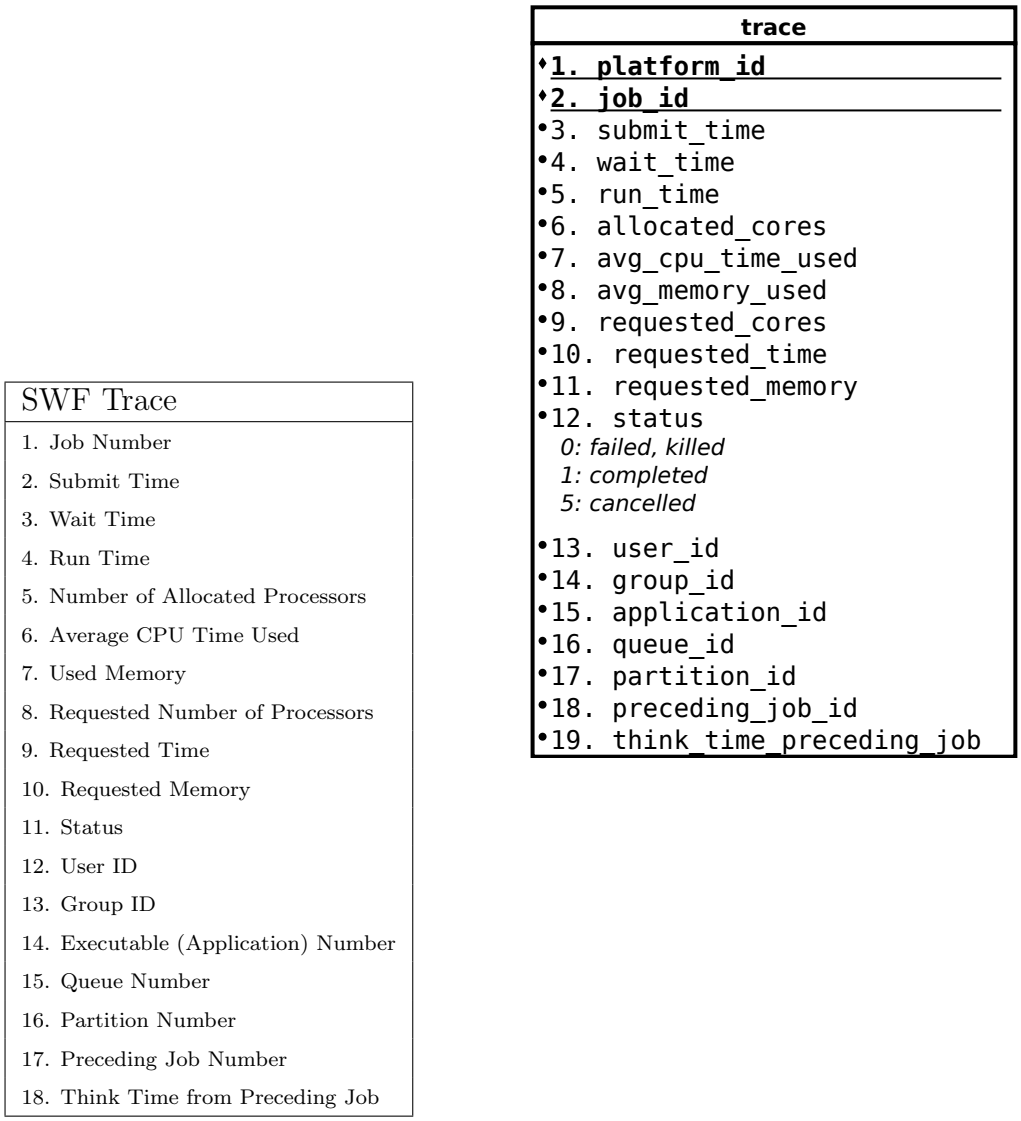


Figure 4.2: SWF Trace and its Conversion to Database Format.

Remark: In SWF, for jobs that have been checkpointed, the same *job_id* might be used in several contiguous lines to describe the different states of the job. Instead this, we propose to use the table *job_events* with the field *event_type* for this description (see SWF extensions). With this choice, the information will be easier to retrieve as this will not split a job in several sub-parts in the trace. This is also more consistent as the checkpoint is not a characteristic of the job itself but a consequence of the scheduling with checkpointing.

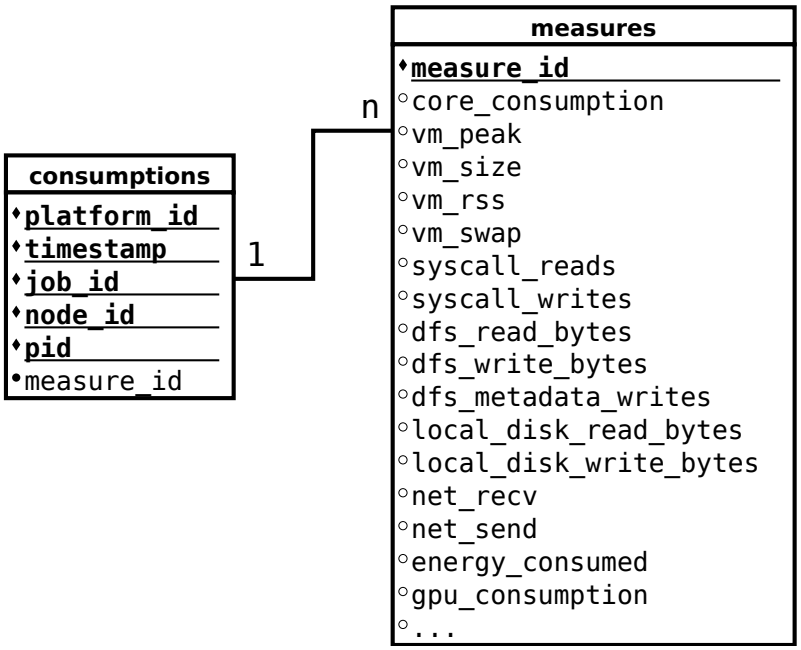


Figure 4.3: SWF Extension: Resources Consumptions.

SWF Extensions

In Figure 4.3 we propose a first extension of SWF that concerns resource consumptions by the jobs. We followed the approach of data storage of the consumption measures as used in Section 3.3. We have a first table *consumptions* that will link a given measure to a job and an other table *measures* that will contain all the metrics collected in a measure. We first start by describing the measures table.

In this table, a measure is identified by the field *measure_id*. This can be an auto-increment, a hash or any other type, but needs to be unique. For a given measure we propose to give several basic metrics. However the idea of this format is to enable users to override it, if a metric is not proposed here, it can be added without problem. When using a database format, the overriding of information does not break the compatibility with the tools made for data analysis. These will simply not request for the added value to the database. Thus, the format allows free customization of the data. Concerning the values of the basic metrics in the measures table, we propose to use system counters whenever it is possible. In its GCT format, Google proposed to use averages on measurement periods. The main problem with averages is the rounding. It is very rare that an average physical resource consumption between two timestamps belongs to $\mathbb{N}$. When we experimented our first analysis of clusters resource consumption in [36] we had troubles of outliers of global core consumption over 100% because of such rounding. This is specially true if the global average comes from rounding of averages. Thus, to keep as precise as possible we propose to use the system counters.

For the values on memory, system calls, file system reads and writes, network

sent and received data and energy consumed, it is easy to retrieve system counters (if the data is available) or to give a calculated value between two measures. However for core consumption it is more difficult. The best way to give a counter on core consumption would be to give the estimated cpu time given by the system to a given pid. The physical value of the number of cycles used by a process has to be correlated with the cpu frequency of the node. This makes the compute of the counter value more complicated and necessitates additional information on node architecture. This is why we chose to use the system estimated time. Reported to the interval between two measures, this enables us in-fine to get the average core consumption.

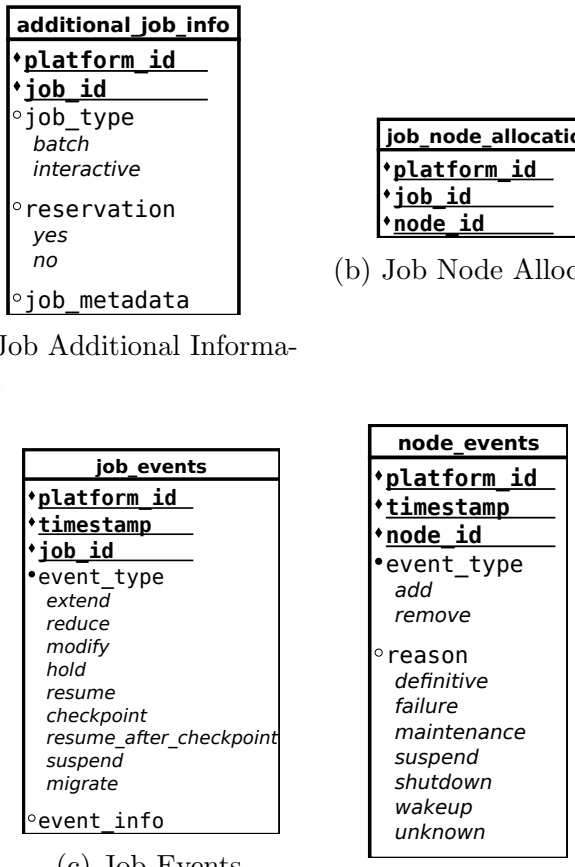
In the table *consumptions* that links measures to a given job, the key is composed of five attributes, and identifies a particular measure. The key attributes are the following:

- *platform_id*, this attribute is used to link a consumption with the trace it comes from. For the analogy, *platform_id* is like the SWF **file**, it refers to a specific trace.
- *timestamp*, a measure is taken at a given timestamp. As we propose to use system counters to provide consumption metric values, a measure will simply give the physical value retrieved by the counter.
- *job_id*, a measure is obviously linked to a particular job.
- *node_id*, we propose to measure consumptions per node, if the measures are an average on the whole nodes allocated to the job, simply put a dummy value here, e.g. -1 to specify that this feature is not used.
- *pid*, same as above, we propose to give on each node values per pid that belong to the job, if *node_id* is set to -1, do the same here.

The way how is constructed the consumptions table enables several levels of granularity in the measures. If a consumption trace contains only averages on all the nodes, only two rows will be added to this table. A first row with a timestamp being the start of the job, and with both *node_id* and *pid* equal to -1. Then a second row with the timestamp being the end date of the job. Thus, in the measures, only two row will be added, the first one with the measure values set to 0 and the second one with the measure values set to the actual value of the counters at the end of the job.

Other types of measures can be added as the ones used in GCT: the number of cycles per instruction or the unmapped pages in cache. This is the great benefit of using a database format, one can augment the consumption metrics according to the needs.

In Figure 4.4 we propose a second extension that provides additional information on the jobs, their resource allocations and node events.



(a) Job Additional Information

(b) Job Node Allocations

(c) Job Events

(d) Node Events

Figure 4.4: SWF Extensions

In Figure 4.4a we propose to dedicate a table to information that is directly missing from SWF. This is the kind of information that could have been added in the trace itself, but we use an other table to really separate SWF from the extensions proposed. The merge of information from the SWF trace and the additional information is simply done by a natural joint of the two tables on the fields *platform_id* and *job_id*. We chose to separate these information from the trace table to allow to convert the trace table to SWF by a simple table dump[1], allowing the easy conversion from the database to SWF. This table adds important data about jobs characteristics not taken into account by SWF as the type of job (batch or interactive) and the information on whether the job was an advance reservation or not. An other field *job_metadata* can also be used to add additional meta information on this job, e.g. if this job was submitted by a human or a robot. This metadata information can also specify if the job was of the type besteffort as proposed by Ciment platform (see Section 3.4). An other example of job metadata would be the resources request as submitted by the user to the RJMS. Although this information is ad-hoc for a particular RJMS, it

[1](but with taking care of removing the *platform_id* field during or after the dump)

is interesting to have the capability to retrieve exactly what the user asked to the system. An informed person would get the plain benefit of such an information.

In Figure 4.4c we propose to give information on jobs events, such as *extend*, *reduce* or *modify* that will be used to describe dynamic jobs [73] modifications of allocated resources and when this occurred. An other type of event can be *hold* or *resume*, that enable to specify when a job was submitted in "Hold" mode and when it was resumed. "Hold" submissions are a particular kind of submission where the job is not scheduled immediately. It will be scheduled at explicit user request; this is called the "resume" of a "Hold" job and has nothing to do with the classical suspend/resume feature used in checkpointing.

Despite the fact that checkpointing information is already included in the SWF trace itself, and thus in the table *trace*, it might also worth to use the event type proposed in the table. Thus information can be defined with a finer grain than in SWF

In Figure 4.4d we propose to give information on the nodes status, as does GCT. For a given node, at a given timestamp we can specify if the node became available or absent from the RJMS. This is described by the field *event_type*. A second field, *reason* will tell the reason of the event. This reason can be a failure, a scheduled maintenance, or a definitive removal. We chose to specify the reason in a separate field because a node can become available or absent for several reasons. e.g. after an absence from a failure, a node can become absent again for a definitive removal if it cannot be repaired. Other types of reasons as suspend or shutdown and wakeup can be used to provide information on the node state if the platform uses energy saving techniques that suspend or shutdown unused nodes.

The database table presented in Figure 4.4b enables to link a particular job to its allocated nodes. This allows to retrieve if a job had to suffer from node failures during its execution.

4.3 Discussion on the Format Proposed

This format proposal aims at being open for customization. Several other types of information can be added such as data coming from the monitoring of the nodes. With this kind of information, it would be possible to verify for a given job, if the system perturbation was important during its execution. The approach of database format enables and eases these customizations.

In the approach proposed we use a database table to store measures data. An other interesting approach would be to use directly in the consumptions table, the HDF5 file of the measure, as provided by Colmet (see Section 3.6).

As we use a database format whose keys are taken from the *job_id* for the tables relative to jobs and from the *node_id* for tables related to nodes, it is always possible to use a subpart of the database format and to use a partial implementation. If jobs resource consumptions are not available, the tables *measures* and *consumptions* will simply not be filled. If node events are not available, it is the corresponding table

that will be empty. In any case, missing information will not prevent from using the schema and basic SWF data will always be available.

Part II

Experimental Evaluation on Resource and Job Management Systems

Chapter 1

Experimentation on Large Scale Platforms

High Performance Computing is characterized by the latest technological evolutions in computing architectures and by the increasing needs of applications for computing power.

A particular middleware called Resource and Job Management System (RJMS) is responsible for delivering computing power to applications. The RJMS plays an important role in the HPC software stack as it stands between the platform, the administrators, the users and their applications.

The experimentation and evaluation of RJMS lead to important complexities due to the inter-dependency of multiple parameters that have to be taken into control.

In [30], D. Feitelson explains the complementarity of the use of models and real workload approaches. Both methods have advantages and drawbacks. We might consider using these two approaches to get a better insight of what is really the system behavior. In [55], Y. Georgiou used modeled workloads within a large scale experimentation approach to evaluate the performance of two RJMS.

In this study we have developed a reproducible methodology and a set of tools to evaluate this kind of middleware based upon controlled experimentation with submission of real workloads under real conditions.

The proposed experimental methodology allows us to obtain large-scale performance evaluation results of production HPC clusters. Our approach is twofold. First, ensure the reproducibility of the experiments by guaranteeing the experiment environment. Then, use a real workload trace extract, replayed in its original context to evaluate a particular RJMS feature, configuration or algorithm. This approach is validated by replaying part of Curie's supercomputer workload with OAR and Slurm Resource and Job Management Systems. The results have been used as insights and allowed us to improve the system's utilization by 19% for OAR and 17% for Slurm considering the specific part of the workload used for the evaluation.

In this part we first explore the context of experimentation with real workloads, then we propose a tool whose goal is to manage the experiments environment and their reproducibility. Then we present our work and results on the evaluation of the

experimentation with real workloads. Finally we discuss the experimental evaluation technique.

1.1 RJMS Background and Research Challenges

Scheduling

The scheduler is a central keystone of the RJMS system. A lot of research with valuable results has been conducted, that compares scheduling algorithms in various contexts [49].

Apart the basic scheduling policies like FIFO (First In First Out), SJF (Shortest Job First) or LJF (Largest Job First), a more sophisticated algorithm that has proven good performance in most of the cases is Backfilling. This algorithm has a lot of variations like conservative, aggressive (EASY) [102] or specific parameter dependent like slack based [110]. In general aggressive backfilling results into better system utilization than conservative [115] but the second one is more widely used thanks to its fairness and guarantee of services.

The parameter of Preemption [8] can also be used for job scheduling to optimize FIFO or Backfilling techniques. The preemption is defined by the stop and later restart of lower priority jobs in order to allow higher priority jobs to perform urgent computations. Another alternative to backfilling policies is proposed by Gang Scheduling. As a subsequent class of co-scheduling, gang scheduling [88] temporarily preempts and then reschedules jobs upon specific time intervals.

The choice of the scheduling policy is an important parameter but it's not the only attribute that has to take all the attention. Various other parameters can play their role on scheduling, depending the RJMS design, its administration and configuration characteristics, the platform architecture and the workload.

A change on the scheduling policy or a specific parameter may provide an important optimization. Scientists need to deal with a big number of trade-offs for all different cases and there is never one best solution to be applied upon all contexts. Particular experimental approaches like the dynP scheduler [107] provide a dynamic policy switching. The basic idea of self-tuning is to generate complete schedules for some policies in each scheduling step. Then these schedules are measured by means of quality metrics like average response and turnaround times and the policy which generates the best result is chosen. In this work we mainly study the scheduling policy of conservative backfilling under similar contexts for OAR and Slurm.

Large-Scale Experimentation Methodology for RJMS

Several studies have been made to evaluate and compare different Resource and Job Management Systems [71], [7], [54]. One of the first studies of performance evaluations for RJMS was presented in [31]. In this study Tarek et al. have constructed a suite of tests consisted by a set of 36 benchmarks belonging to known classes of benchmarks

(NSA HPC, NAS Parallel, UPC[1] and Cryptographic). They have divided the above into four sets comprising short/medium/long and I/O job lists and have measured average throughput, average turn-around time, average response time and utilization for 4 known Resource and Job Management Systems: LSF, Codine, PBS and Condor.

Another empirical study [14] measured throughput with submission of large number of jobs and efficiency of the launcher and scheduler by using a specific ESP benchmark [118]. The study compared the performance of 4 known RJMS: OAR, Torque, Torque+Maui and SGE. ESP benchmark has a lot of similarities with the previously described suite of tests [31]. However one of its advantages is that it can result in a specific metric that reflects the performance of the RJMS under the specific workload and the selected scheduling parameters.

The approach used in this work is an extension of the methodology used in [14], since we extend the use of workload replay as in ESP, but with real workload traces execution upon a production system.

The experimental methodology presented in this work makes use of Grid'5000 platform [15] which is a testbed composed of a collection of various clusters and software tools dedicated to research on computer science. Grid'5000 platform has the advantage of allowing the reconfiguration of environments and reproduction of experiments, hence providing the ideal tool for real-scale experimentation of Resource and Job Management Systems. One of the most important factors for the study of Resource and Job Management Systems, is the users workload which serves as their input. To be able to compare, study and evaluate those systems, we need to define specific workloads and application profiles that are going to be used.

Based on previous work [51], [48] it seems that to give precise, complete and valuable results the studies should include the replay of both modeled and real workload traces because of their complementarity. A real trace will be the most "real" test of the system as it embeds all the complexity of the underlying workload, however it represents a specific workload on a given platform with its characteristics[2] and may lack of generality. This can be provided by a model. In previous study [55], modeled workloads were used to evaluate an RJMS system, whereas in this work we focus on real workload deployment.

1.2 Using Grid'5000 as a RJMS Evaluation Platform

The Grid'5000 [15][3] experimental platform is a scientific instrument designed to support and ease experiment-driven research on distributed systems at large scale. Similar to projects like FutureGrid[114][4] and PRObE[5], Grid'5000 provides a highly

[1]UPC (Unified Parallel C): http://upc.lbl.gov/

[2]these characteristics are numerous: configuration, administration, users, constraints

[3]https://www.grid5000.fr

[4]https://portal.futuregrid.org/

[5]http://www.nmc-probe.org/

reconfigurable easily accessible and controllable environment that researchers can use to perform experiments that are not possible at a smaller scale. With 9 sites in France, one other site to come and 1 in Luxembourg, Grid'5000 allows its user to experiment at both cluster and grid level. Each site is interconnected into a single grid enabling users to experiment at grid level. Each local site is composed of one or several clusters enabling users to experiment at cluster level. To use the platform, the researcher reserves a partition of the cluster or grid that will be fully dedicated to the experiment. With such a platform, it is possible to observe at scale and study deeply distributed systems behavior under real life conditions. Cluster and Grid systems are complex and their study need the full control of all the layers of the software stack between the user and the hardware. The applications, programming environments, runtime systems, middlewares, operating systems and networking layers are subject to extensive studies seeking to improve their performance, security, fairness, robustness and quality of service. Thus, a mechanism that would facilitate the experimentation upon all different layers of the software stack, became indispensable. The ability to easily control, reconfigure and monitor the full software stack makes of Grid'5000 a powerful tool for complex distributed systems evaluation and experimentation. At reservation time, users can specify if they require a normal experiment run as in standard clusters or grids, or a "controlled" experiment run. This "controlled" experiment run is called *deploy* mode and uses the Kadeploy tool.

The Kadeploy [56, 57][6] tool, available on the platform, enables Grid'5000 users to take the full control of their experiments by allowing them to configure, deploy, install, boot and run their specific software environments including all the layers of the software stack. This enable a deep reconfiguration of the platform by the users. In order to do this, users have to respect the following process: first, reserve a set of nodes (this might be at cluster or grid level), deploy an environment then log as root on the allocated machines. The environment deployed can be a default production environment, like the one available in the normal mode or a custom environment corresponding to their needs.

To deploy the environment on a reserved node, Kadeploy first "pushes" the software environment on a dedicated disk partition of the node then reboot the node on this partition. To do this it uses traditional protocols for network booting: PXE, TFTP, DHCP and connects the nodes through SSH. After the experiment has run or after an experiment crash, Kadeploy will reset the node in its original configuration with the default production environment image.

We use the Grid'5000 platform for our experiments on RJMS evaluation. The ability to configure the reserved nodes enables us to setup an environment image that embeds all the software, configuration and tools we need for our experiments. Thus the experiment protocol is simplified: reserve a set of nodes; deploy the environment image on this set; deploy the RJMS infrastructure with our automatic tools installed on the environment; run the experiment. Thus basically, all we need to run the experiment is a set of nodes and our customized environment. The main work thus is

[6] http://gforge.inria.fr/projects/kadeploy

in the construction of this environment. On Grid'5000, an environment is created very simply by making an archive of the root partition in compressed tar format. However, the sole environment image is not sufficient to control and keep a trace of how the environment was constructed. We need a way to describe the process of the configuration on the different software parts on the experiment image in order to later understand how it was conducted, to ease the update and to help for the environment portability (e.g. on other research platforms). A tool named *Kameleon* has been developed for this purpose and to ensure experiments reproducibility, it is presented in Chapter 2.

The experiments reproducibility is important to obtain reliable observations; with the use of Kameleon and Grid'5000, we performed our experiments multiple times to guarantee the validity of our results.

Chapter 2

Reproducibility

In the scientific experimentation process, an experiment result needs to be analyzed and compared with several others, potentially obtained in different conditions. Thus, the experimenter needs to be able to redo the experiment. Several tools are dedicated to the control of the experiment input parameters and the experiment replay. In parallel concurrent and distributed systems, experiment conditions are not only restricted to the input parameters, but also to the software environment in which the experiment was carried out. It is therefore essential to be able to reconstruct this type of environment. The task can quickly become complex for experimenters, particularly on research platforms dedicated to scientific experimentation, where both hardware and software are in constant rapid evolution. This work discusses the concept of the reconstructability of software environments and proposes a tool for dealing with this problem.

2.1 Introduction and Motivations

Research platforms dedicated to experimentation, like Emulab[1], Grid'5000 [15] and PlanetLab [18], offer to their users mechanisms allowing them to deploy and run their experiment on a set of nodes. In infrastructures such as Grid'5000, users can deploy their own software environment in which the experiment will be executed. This versatility gives experimenters a great flexibility but it is usually expensive. The user has to manage both his application and the environment in which it will be executed. In such a scientific experimentation context, it is necessary to ensure experiments results reproducibility. The reproducibility of an experiment is the ability for a researcher to reproduce the original experiment (that may be his or another one's). It is slightly different from the repeatability as it does not involve a strict exact same conditions to redo the experiment[2]. The reproducibility of the results is thus the ability for the researcher for a given experiment, run several times (several instances) in the same conditions, to give the same result each time, or at least a

[1] http://www.emulab.net/pubs.php3

[2] repeatability needs the same observer, procedure, measurement techniques

result statistically comparable with small standard deviation. It is commonly agreed that below a probability of 0.95 for the difference between two test results obtained under reproducibility conditions is over the acceptable limit. Without the results reproducibility, the experimenter cannot compare experiments [59, 2]. Indeed, it is not acceptable to draw a conclusion based on non-reproducible results. Thus, the experiment in itself has to be reproducible and it is essential to be able to recreate its observation conditions because they will determine the results. These conditions are:

- the experiment instances parameters: their input data and configuration settings;
- the environment in which the experiment will be executed: software and hardware (taking into account its state at the run time).

The different parameters can be retrieved by using an experimentation plan, that log all the parameters used along the experiment instances, or by using an experiment conduct tool. The hardware environment condition requires that the experiment has to be run on the same set of nodes each time or at least on a set of nodes with identical characteristics, but the problem of the hardware qualification before the experiment still remains. The user has to be sure that the same hardware is in the same state for each experiment. The reconstruction of the software environment is also a problem as it might be a difficult task to do. This environment reconstructability problem will be the main topic of this chapter.

For A.Iosup in [64], experiments reproducibility has two conditions: ensuring that the experiment can be run in the same conditions each time, and ensuring that the provenance of data used in the experiment (where data comes from) is known and that data can always be retrieved. Data provenance is a common problem in computing science [101]. More particularly in domains related to computing platforms such as Databases [11, 10] and Grid Services [108]. This work proposes to extend the data provenance concept by considering also data that is used to build the experiment software environment. This is justified by the fact that data (software, versions, configurations, and files) that make an environment determine some part of it and are necessary for its reconstruction, and so, to the results reproducibility of the experiment [59]. This work proposes to link data provenance to reconstructability.

This work has presented the software environment reconstructability as an important part of results reproducibility. Indeed, software updates, hardware evolutions, platform reconfigurations damage software environment reconstruction and experiment reproducibility. Thus, it is difficult to make a software environment durable. A tool is necessary to solve this problem.

This work is organized as follows, in Section 2 we give more details on why it is necessary to be able to reconstruct a software environment and how data provenance has to be considered in this process. In Section 3, we introduce Kameleon, a tool that aims at easing the environment reconstruction. Then, in Section 4 we present the validation of our tool on the Grid'5000 platform through use cases taken from

real users' experiences. In Section 5, we present a short state of the art of related tools. Finally in Section 6 we conclude and state some evolutions perspectives.

2.2 Problem Details and Background

In the experimentation process, the different instances of an experiment have to be run in the same software environment to ensure that the conditions are the same, thus allowing results comparison. Or so, the experimenter has to be able to know what changed and to control these changes between each run. Indeed, not all the software stack of an environment is implied in the result but only a subpart, depending on the experiment itself. If we ensure that the software part which determines the experiment behavior and thus its result is common to all of its instances, the results can be processed. Moreover, controlling what changed between several environments may be wanted by the experimenter for comparison purpose. He may want to create several versions of an environment, with different versions of a particular library, software or configuration and observe how his experiment behaves in this environment. Thus we have to be sure that the only differences between the environments were the ones wanted. In the research platforms dedicated to experimentation, both hardware and software are in constant rapid evolution. Thus, a software environment that is currently working and up to date may be obsolete very quickly, thus backing up the software environment and restoring it before each experiment run is not sufficient. The user custom software environment can become useless and will have to be created again, from scratch. In this case, the experimenter must have kept a trace of the environment construction to be able to reconstruct a new working environment, similar to the first version.

Environment Construction Steps

Generally, constructing an experiment software environment is composed of many short operations (usually shell commands), that are not linearly distributed over time. At the beginning, the user sets up the environment "basis" with a set of commands. This "basis" will be all the software stack needed by the experiment to run properly. E.g. the Operating System (OS) that will be installed on the nodes running the experiment, software required that have to be installed, their configurations, network and users setup. All these "basis" set up commands can be easily merged into one single script destined to reconstruct the environment later. However, this is generally not enough and requires several debug and setup stages. During these phases, the basis environment is modified by small patches and their integration in the global script gives a result hard to maintain and impossible to be reused by the users' community. Moreover, this method has the disadvantage of being less robust to errors. The risk of unlogged modifications to the environment, not reported in the final script, is significant. Another drawback of this method is the lack of flexibility. The environment has to be generated directly on one of the nodes of the experimentation

platform and thus monopolizes one node, generally for several hours just for the construction purpose.

Data Provenance

When reconstructing a duplicate environment, it is necessary to look at two matters. First, the environment creation process, i.e. the different steps that lead to the environment generation. These steps are the set of commands and related input data that allowed to setup the software environment and its configuration. Second, data contained in the environment have to be considered too. The reconstructed environment must contain the same data as the original one. This idea of data has to be as large as possible and has to take into account the files that can be present on the environment, but also installed software and their versions. This idea of data provenance, central in the reconstructability context, states the problem of software changes and evolutions in the packages repositories or direct download links. One cannot guarantee that a particular download link will exist forever or always point to the same software with the same version and build tag. It is the same thing for packages repositories whose software versions evolve along the updates. If the packages repositories are updated between two environment generations, the two environments will inevitably be different, and the experiment reproducibility may be affected. There is thus a need of a method that guarantees that both original and duplicated environments use the same data sources. That is what was previously identified as a data provenance problem.

Features Useful for Reconstruction

The reconstructability concept should also answer to several more technical problems, such as:

- **Debugging**. The availability to manually operate whenever during the environment creation has to be eased and easily traceable.

- **Composition of several environments parts.** To pool efforts, it should be possible for the users to share the whole or some parts of their environments to enable the experimenters' community to reuse them.

- **"Snapshotting"**, or restart after backup. Developing a software environment is rarely a simple and linear process. The possibility to restart from a former existing environment is a feature which brings a lot of comfort and saves a considerable amount of time to the user.

2.3 A Tool for Reconstructability: Kameleon

General Principle

Kameleon[3] is a tool, written in Ruby (for the engine)[4] and Shell (for the environments construction)[5] that has been developed to answer the problem of software environment reconstruction. The general principle is simple: the Kameleon engine enables composing chunks of scripts, boosted with special commands. Their goal is to mask some technical complexities to the user. Kameleon can thus be seen as a command sequencer, supplied with tools to ease some operations. During an environment (re-)construction, Kameleon will read the environment description and the stages that make up its creation in a "recipe" file described in YAML[6], a human friendly data serialization standard for all programming languages. YAML has been chosen due to its user-friendliness, its simplicity, and because it is implemented in several programming languages, easing the coupling between Kameleon and higher level tools. The "recipe" is made of two parts. First, the global variables definition part. Second, an ordered list, detailing the different steps of the environment creation. Each item in the list corresponds to a YAML "step" file, whose name will be unique in the whole recipe. Each step corresponds to one semantic action like installing a software, modifying a particular configuration file or recording the environment as a particular output format. A step is thus a set of commands leading to one or several technical actions. The sum of all the steps that compose the recipe will generate the software environment.

Inside a step, it is possible to define subparts. They will be used in the recipe, instead of calling the full step, the user can suffix the name of the step with one or several of its subparts, thus only these ones will be taken into account in the environment construction. The global variables defined in the recipe can also be accessed in the steps (by preceding their names with "$$").

In the global variables definition part of the recipe, a particular field is mandatory and reserved for Kameleon: "distrib". This field will be used by Kameleon to know which kind of OS will be setup in the output environment (e.g. debian, fedora, SL5 are possible values). As the different OS distributions can be very different, installation and configuration steps of a particular software may differ according to the OS type. Kameleon will use the steps corresponding to the right OS during the environment creation. Thus, with Kameleon, a recipe that creates a Debian virtual image can be easily transformed into one that creates a Fedora virtual image. The changes to make in the recipe would be to replace the "distrib" dedicated field in the header of the recipe from Debian to Fedora. The result will be an environment with the same software and configurations but whose OS will be Fedora.

The goal of the slicing between recipe and steps is to ease the code reuse and

[3]http://wiki-oar.imag.fr/index.php/Kameleon (Work In Progress)
[4]for about 800 Lines of Ruby Code
[5]for about 3000 Lines of Shell Code
[6]YAML Ain't Markup Language, see http://www.yaml.org

distribution. A step, once tested and validated, will be used in several recipes and by several users. The idea is that it is more efficient in a developers community, to pool their qualities, and that everyone works on a subpart of the problem, in his competence field. Kameleon tries to encourage this approach.

An other benefit of this approach is the ease of the process of generating several environments "flavors". If the experimenter needs three environments with three different versions of his application (or a particular library), he will create three different steps corresponding to the installation/configuration of these three versions. When generating the environments, just setting the corresponding step in the "recipe" file will allow Kameleon to generate the environment with the wanted application version. Thus the result will be three environments whose only differences are the versions of the experimenter's application.

An example of a Kameleon recipe and some of its steps is presented in Figures 2.1a and 2.1b. In the "recipe example" presented, highlighted steps are the ones shown in the "steps example". Here, the recipe will generate a basic "debian squeeze" environment in the "raw" output format (a generic format which is directly exploitable into a virtual machine manager or that can be converted later). Concerning the two highlighted steps, the "bootstrap" step will create the basis debian installation through the "debootstrap" command, which is a common method for installing a debian system. The "kernel-install" step will write a kernel configuration file, then install it.

The whole environment generation process using the recipe and steps files is shown in Figure 2.2. The Kameleon engine loads a recipe file and parses it to retrieve the step files called in the recipe. Then, the steps are processed in their order of appearance in the recipe. Finally, Kameleon will produce as output the environment generated and a cache area containing data used to build the environment. The reason of the presence of the cache area will be explained later in this work.

With such a method, the only modifications that have to be done in this recipe are changing the system installation part from Debian to Fedora. This is done by replacing in the recipe the few steps responsible for the OS installation from Debian to Fedora (both are provided with Kameleon). In our example of modifying the recipe to get a Fedora instead of a Debian distribution, the changes to make to the recipe would be: replacing the "distrib" field located in the "global" variables part, in the header of the recipe from debian to fedora. This will make Kameleon use the Fedora "`bootstrap`" and "`kernel-install`" steps that are specific to the system installation instead of the Debian ones. Basically, just replacing "debian" to "fedora" in the recipe and running Kameleon with this recipe would create a second image with the same software installed and in the same output format. When setting the "distrib" field in the recipe, the user tells to Kameleon where to find the steps to use in the steps hierarchy. Thus, Kameleon will automatically load the steps corresponding to the distribution specified in the header of the recipe and will execute them.

Recipe sample: Creation of a KVM Debian image

```
# ### debian.yaml Kameleon recipe sample ###
global:
  distrib: debian
  # Debian specific
  debian_version_name: squeeze
  distrib_repository: http://ftp.fr.debian.org/debian/
  #
  # Architecture
  arch: amd64
  kernel_arch: "amd64"
  #
steps:
  - bootstrap
  - system_config
  - root_passwd
  - mount_proc
  - kernel_install
  - strip
  - umount_proc
  - build_appliance:
    - clean_udev
    - create_raw_image
    - create_nbd_device
    - mkfs
    - mount_image
    - copy_system_tree
    - install_grub
    - umount_image
    - save_as_raw
  - clean
```

(a) Kameleon recipe example.

Debian Kernel installation step

```
kernel_install:
 - kernel-img_conf:
   - write_file:
     - /etc/kernel-img.conf
     - |
       do_symlinks = yes
       relative_links = yes
       do_bootloader = yes
       do_bootfloppy = no
       do_initrd = yes
       link_in_boot = no
 - kernel_install:
   - exec_chroot: bash -c "DEBIAN_FRONTEND
     =noninteractive apt-get -y --force-yes
     install linux-image-$$kernel_arch"
```

Debian basis installation step

```
bootstrap:
 - debootstrap:
   - exec_appliance:  debootstrap --arch=
     $$arch $$debian_version_name
     $$chroot/ $$distrib_repository
```

(b) Kameleon steps example.

Figure 2.1: Kameleon recipe and steps example.

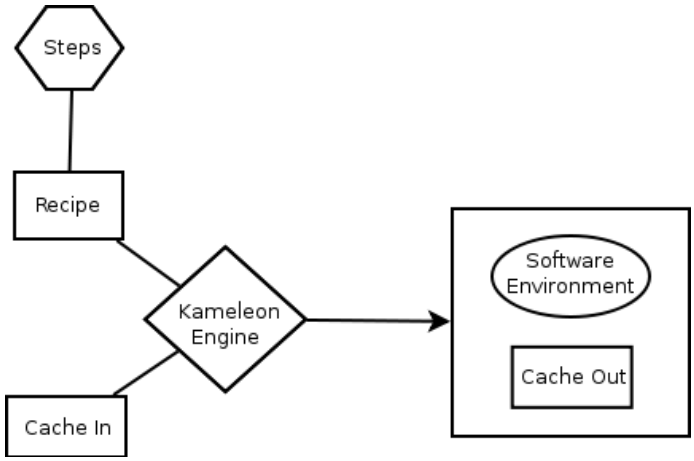


Figure 2.2: Environment generation with Kameleon.

Execution Contexts

Kameleon steps are composed of a set of shell commands prefixed with specific "Kameleon-commands". Their role is multiple.

Some of the "Kameleon-commands" enable the Kameleon engine to know in which context the corresponding action has to be executed. This notion of context is important here. It illustrates the isolation between the machine running Kameleon (host) and the environment being created by Kameleon. Depending on the goal of the step, the action to execute can be run in the environment itself or in the context of the machine that runs Kameleon. A typical example of the first case is a software installation. If the action is the installation of a particular software in the environment, the installation context has to be the environment and not the machine running Kameleon. Indeed, the goal here is certainly not installing the software on the host. An example for the second case is the following: a file has to be copied from the host File System to the environment. Here, the copy of the file has to be done in the context of the host in order to have access to the file. Due to the isolation mechanism between the host and the environment, the environment cannot access the host File System. The Kameleon tool uses the "chroot" isolation method (system root change) for all the actions that have to be executed in the environment context. This method has been chosen because it is a common technique for Unix distributions installation bootstrap.

The others special commands provide abstraction methods to elementary operations. They allow to:

- include another step file. Everything described in the included step will be executed;
- verify the presence of a particular command (in the environment or in the host context);
- manipulate files (append, write);
- define local variables. Instead of the ones defined in the recipe, that are global, these have only a local (step-level) scope;
- get a shell (with the possibility to invoke a particular step by its name).

Provided Ingredients

Kameleon is distributed with a set of recipes and steps. Their goal is to provide basic bricks and examples for the user and help him developing his owns. Among provided steps, Kameleon offers environment export methods in several output formats. Currently, the output formats are VirtualBox[7], Xen[8], KVM[9], disk image, Grid'5000 image and archive. This allows the user to generate his environment on the format he needs, and thus, to operate the environment creation on an other machine than the one dedicated to the experiment.

[7] http://www.virtualbox.org
[8] http://www.xen.org
[9] http://www.linux-kvm.org

A virtual appliance, configured to use Kameleon, is available for download on the official website. A Kameleon recipe is dedicated to generate this appliance.
Kameleon also provides a recipe and its steps to generate a Debian environment. RPM packages based distributions are under development.
The goal of providing all these "ingredients" with Kameleon is to enable the user to easily reuse what have been done by the community. A dedicated repository is available to share the steps and recipes between Kameleon users. Users are obviously welcome to submit their own steps and recipes for them to be integrated in the repository. Of course, a lot of steps are very specific to a particular platform or OS but this problem is inherent to dealing with multi-platforms and OSes during the experiments and no simple method will work for all of them. The most convenient, maintainable and comprehensive method is thus to provide many little specific basic bricks, easily adaptable for particular cases. Thus, the Kameleon tool helps a lot in organizing the environment construction. Besides, all the steps that are not platform specific can be directly used in several recipes. This is a substantial saving in terms of time.

Snapshotting

Kameleon provides a way to use recipe snapshotting (i.e. restart of an environment construction after backup). In the recipe, it is possible to specify an archive, that will be used as a base for the environment construction. This technique is particularly used during the iterative phase of the recipe/steps development. Instead of restarting from scratch each time, Kameleon reuses what has been validated as working, and being backup. The Kameleon engine loads the snapshot archive and will set it as the new environment context (in the chroot). Figure 2.3 illustrates how snapshotting can be used to shorten the image generation process by reusing what is known as working. In this example, a first environment "E1" has been built with Kameleon and the "Recipe 1". Then, from this environment archive, the user can create a second "E2" environment, based on "E1". The new environment will contain everything that was in "E1" plus what was described in the "Recipe 2".

Data Provenance

Later Kameleon developments tends to bring an answer to the data provenance problem. In the recipe, Kameleon allows to specify a cache area and an associated policy. This policy will allow the user to define if data accessed during the environment creation will come from the cache, from outside, or both. Thus, after the environment generation, Kameleon will have in its output zone on the machine that ran the recipe: the environment itself and the cache being used for its creation as shown in Figure 2.2.

Kameleon provides two methods to use this caching idea. First, at launching, the tool configures an HTTP proxy to cache all the files retrieved through this protocol (which represents most of outside access in a common Kameleon run). Thus, at the end of the Kameleon run, every data accessed through HTTP during the environment

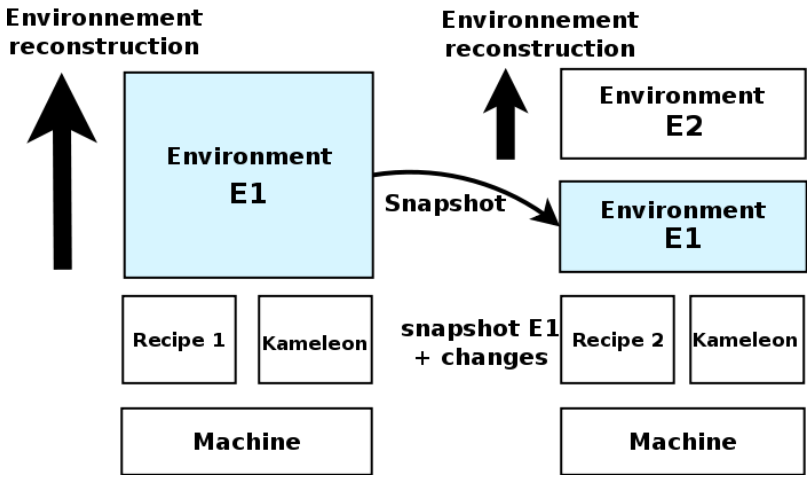


Figure 2.3: Kameleon snapshot feature.

creation will be available in the output cache zone. For other protocols, a command allows to specify a file and the way to retrieve it, if not present in the cache. When this command is called in a step, data will be either read from the cache or retrieved from the outside and then fill the cache (according to the cache policy and contents). So basically, it is possible to cache every data that have been used in the environment construction. The user will decide which data is important for the reconstruction and thus needs caching.
An interesting problem here, is data with restricted distribution (data that should not be distributed for various reasons like private or classified data). Such data retrieved during the environment creation can end into the cache automatically. This is a problem because when the user will want to redistribute his environment and its associated cache, the data that should be considered as "private" (e.g. ssh keys) will be found in the environment cache. To solve this problem, Kameleon offers to specify it as a parameter in the previously described command, and will split the cache in two parts: "public" and "private". The user will thus decide which data is "private" and which is "public". Kameleon offers a method allowing to export the whole recipe, steps, cache and environment generated, for redistribution. This takes into account the private data and, instead of exporting data itself, only the data fingerprint and metadata will be exported. This allows to know if data that composed the environments during the different experiments was the same.

2.4 Use Cases

All the cases presented here are taken from **real users experiences** on Grid'5000 (in short: G5K). This platform has been chosen to validate our approach for its flexibility and its richness. An advanced use of Grid'5000 often requires from the users to deploy their own software environment on a set of nodes. Such a situation

often occurs and thus illustrates the gain in using Kameleon for a repetitive action. Grid'5000 provides a wide range of use cases, allowing to illustrate several points of interest of Kameleon. However, Kameleon is absolutely not linked to Grid'5000 in any way and aims at being used on any system where the user can install his own software stack, from the simple desktop machine to the reconfigurable computing platform or even on embedded systems. The use of Grid'5000 in this section is only made for validating the Kameleon approach.
Grid'5000 is a scientific tool dedicated to the study of large scale parallel and distributed systems. Geographically distributed over 9 sites in France, one to come and 1 in Luxembourg, it aims at providing a highly reconfigurable, controllable and monitorable experimental platform to its users. This platform has the particularity to allow them to deploy their own software environment, which makes it perfectly adapted to the Kameleon use.

Figure 2.4 presents the environment deployment process on Grid'5000 (here, we do not need Kameleon, this figure only illustrates the deployment process). The user reserves a node, once it is allocated the software environment is sent to the node through a specific tool. Basically, this tool copies the environment archive to a dedicated disk partition, which is bootable and then the node reboots on this partition. Once this is done, the user can connect the deployed node through ssh with the root account. Several default environments are available on Grid'5000 and can be deployed. The user can both deploy his own environment or a default one.

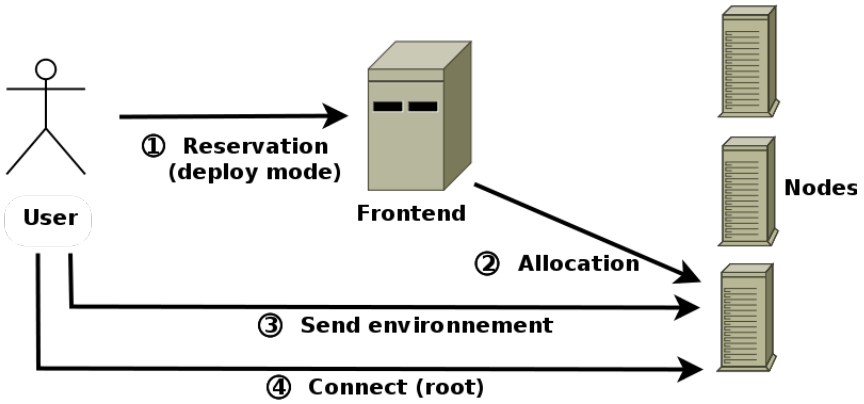


Figure 2.4: Environment deployment on Grid'5000.

Figure 2.5 shows the typical usage of a deployed environment customization on G5K without the use of Kameleon. Once the environment is deployed on the node and the node has rebooted on this environment, the user connects to the node and can modify the deployed environment. When the modifications are terminated, the new environment has to be saved for a further deployment.

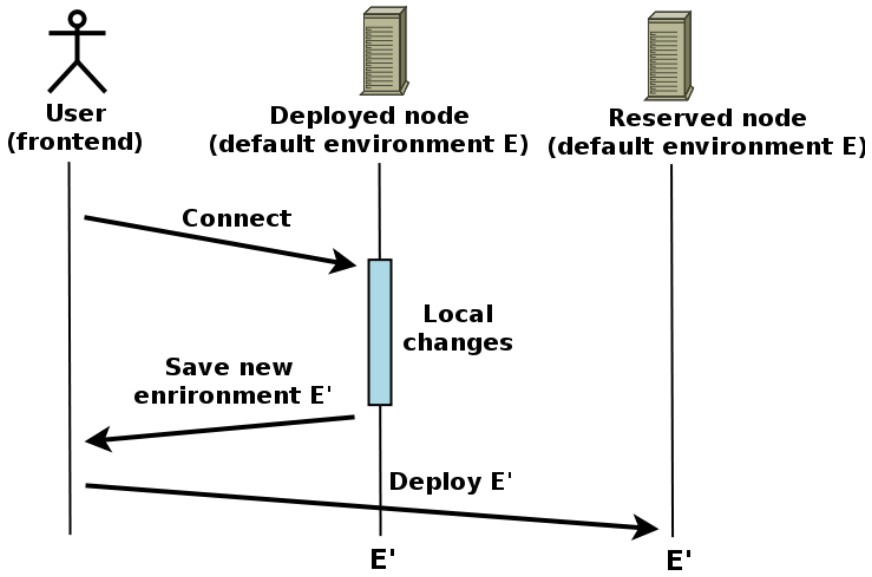


Figure 2.5: Regular environment customization on Grid'5000.

In Figure 2.6, the process is completely different. The user starts from a default environment already available for deployments on Grid'5000, loads it thanks to the Kameleon snapshot feature and then modifies it locally before saving it. The great benefit here, is that the user does not need to reserve a node of the platform for the environment customization but does it locally, on his workstation and the platform is not required during this phase.

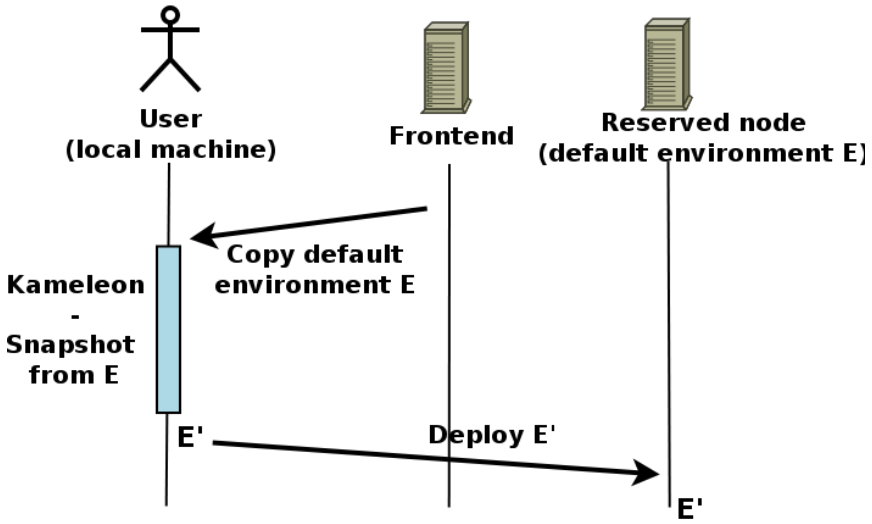


Figure 2.6: Environment customization on Grid'5000 with Kameleon.

Figure 2.7 shows another different use case: here the user creates from scratch his own environment, and then saves it for a later deployment. This might be very useful if the user needs another Operating System (OS) that the ones provided for deployment. E.g. on Grid'5000 there are no base images provided with the SL5 (Scientific Linux 5.0) OS. This OS is required for installing a gLite[10] grid middleware. Thus for testing purpose of the gLite software, the user might need Kameleon to construct from scratch the SL5 OS image required.

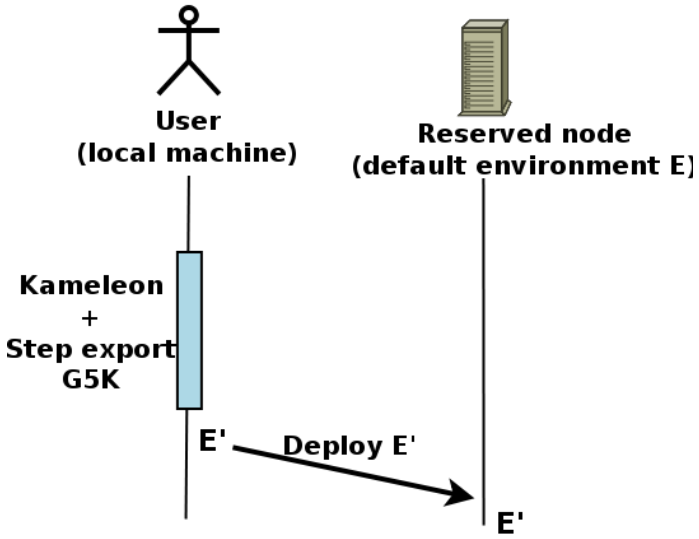


Figure 2.7: Environment creation on Grid'5000 with Kameleon.

Figures 2.8 and 2.9 present a typical usage where the Kameleon tool allows the user to take control over what changes between two environment reconstructions. Let's take the case of a platform upgrade. New network cards are installed in the nodes, providing an other type of network (i.e. Infiniband) to the users. In such case, the deployable environments provided by the G5K technical team are changed: a new kernel is required to support the new device. These environments are reconstructed by the administrators, leading to a set of complete new environments with new software versions.

In Figure 2.8, the user has customized, built and installed several software and saved his own environment. He got several results from his experiments. When the platform is upgraded, the new network cards are not supported by the old kernel of the user's environment. His custom environment has no network access and thus is unusable. The user will have to recreate it from the new updated environment provided by the technical team. He has to reconfigure it and spend a lot of time to make it work as previously. But worst, now his new custom environment is different from his former one. Software versions differ (and particularly gcc and all

[10]Middleware for Grid Computing (http://glite.cern.ch)

the compiling tools) and he might have troubles when building the softwares his experiments requires. Moreover, the user is not totally sure about the validity of the comparison between his previous experiments (in the first custom environment) and the ones done in the new environment, the installed software having different versions.

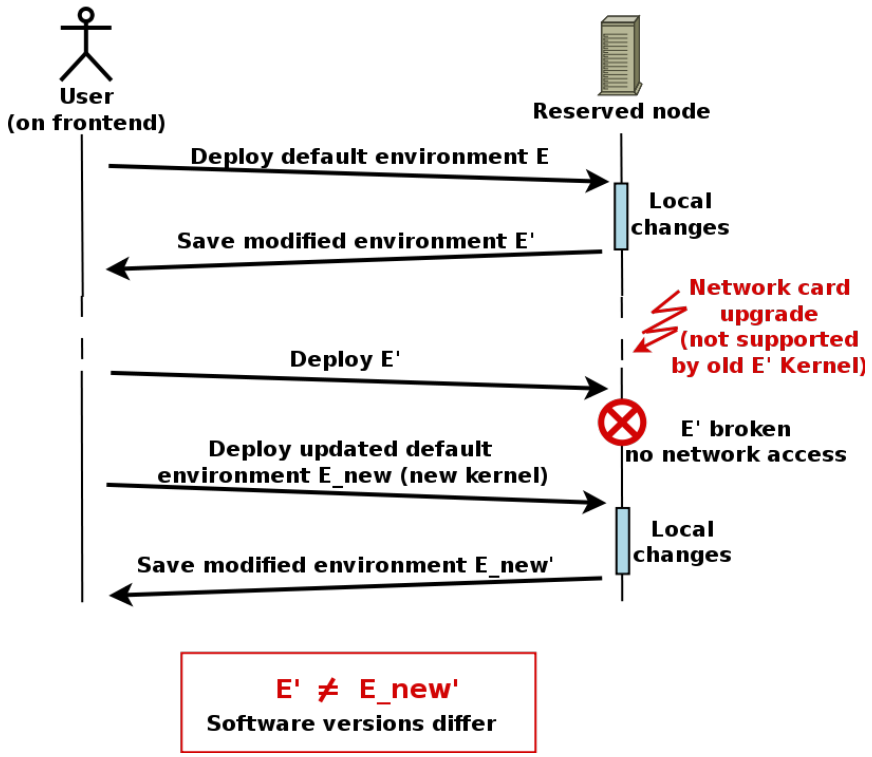


Figure 2.8: User Environment aging problem – without Kameleon.

In such a situation, Kameleon is very helpful. Figure 2.9 shows how a user can take advantage from the Kameleon snapshot mechanism. Here, the user has customized his environment with Kameleon from a Grid'5000 default environment. He has loaded this default environment in Kameleon, modified it according to his needs and saved it. After the network cards upgrade on the machines, the user will be able to update locally (on his own computer) only the kernel part of his environment by loading it in Kameleon and applying it a recipe that upgrades the kernel. Thus, with this solution, the new environment can access the network and only the kernel will differ between the original environment and the new updated one. The results obtained with the old environment have less chances of being trashed because of software version incompatibility.

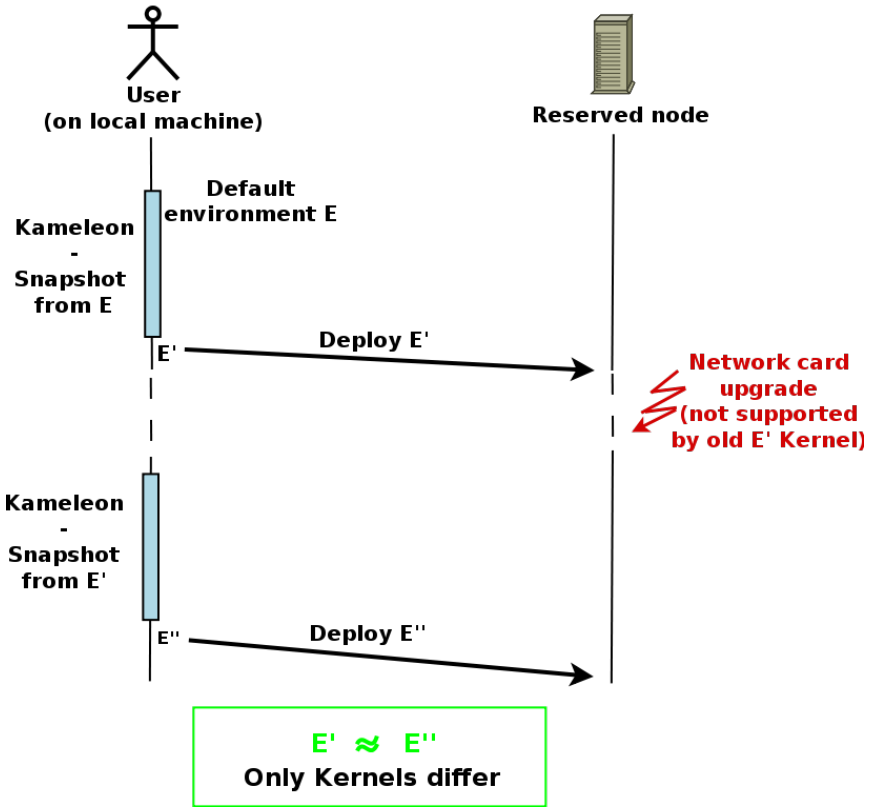


Figure 2.9: User Environment aging problem – with Kameleon.

2.5 Related Works

Several configuration automation and systems administration software currently exist. Among the most recognized, there are CFEngine[11], Puppet[12], and Chef[13]. Historically, Puppet came after CFEngine and has been strongly inspired from it. More recently, Chef takes up their concepts and offers an Open Source solution. Their goal is to ease the setup, the administration and the maintenance of an infrastructure. These 3 tools use a Domain-Specific Language (DSL)[62], a dedicated language, relative to an application domain, to describe an environment software configuration. This type of language allows to express, in an abstract way, thanks to an executable language, all the semantic of the domain it refers to. This is both an advantage because it allows to define simple programs, easy to understand, but also a major drawback, the effort to master it at start, is considerable. Thanks to their DSL, these tools allow the infrastructures administrators, for a given machine (or type of machine), to describe which applications and services will be present and what will

[11]http://www.cfengine.org
[12]http://www.puppetlabs.com
[13]http://www.opscode.com/chef

be their configurations. Then, the automation software engine will allow to install and configure this machine, as well as managing its potential software updates. For more information concerning the use of a DSL in these tools, a substantial study is presented in the technical report [60]. A state of the art and a global comparative study of such tools is proposed in [24]. These tools allow to answer partially to the experiment environment reconstructability problem. Indeed, they propose to describe, in a machine administrator point of view, a software environment. However, the real goal of these tools is not reconstructability. Thus, they are not really convenient in that case. The use of a DSL restricts the versatility, what can be expressed is determined by the tool itself and its DSL. These tools are focused on "configuration" for administration, and not for experimentation. Being designed to manage a platform and not generate software environments from scratch, the system creation bootstrap is a problem. What can be done in such a case, is the environment configuration part once the basis system is already installed. The user has to initiate this part in an other manner, manually or by a script, but in a external way. Moreover, these tools are quite heavy, in terms of infrastructure, to set up, and work on the Client/Server mode. Only Chef proposes a lighter version: Chef-Solo, which is autonomous. Finally, the data provenance problem is not taken into account with such solutions.

A more interesting tool, which is closer to our problem is CDE [3]. The idea is to allow users to eliminate the dependencies problem during the installation of a software or a library. To do this, CDE proposes to catch on the fly all the accessed files (binaries, libraries, configuration files) during a command execution, and to provide this set as a CDE package. This package can be redistributed and run on an other machine, as long as it has the same architecture (e.g. x86) and the same major kernel version (e.g. 2.6.X). More complete than the static links mechanism, which only allows a program to be linked to its libraries, this tool allows to capture all the environment necessary to the execution of the command. In fact, related to the reconstructability problem, it can be said that this tool captures the state of the software before the experiment. This solution is incomplete for the problem since it does not capture all the experiment environment, but proposes an interesting approach. A CDE archive is more portable than a full system archive (as in Grid'5000) which is dependent on the physical machine construction (i.e. disks partitions, network cards configurations), it only depends on the kernel system calls. But with this solution, the construction plan of the environment is not kept as it is in Kameleon, with the recipe and steps. Moreover, the CDE mechanism does not allow to treat fully the data provenance problem neither.

2.6 Conclusion and Perspectives

This work stated one of the major problems encountered during scientific experimentation, and more particularly in the research platforms dedicated to this domain: experiments and results reproducibility. This work explained the reproducibility conditions, and identified a particular notion related to reproducibility: the software

environment reconstructability concept. This work combined to this the idea of data provenance, an essential counterpart to experiments reproducibility. Then, a tool to solve these problems is proposed and some use cases of this tool are given. Finally, this work stated several tools that can be possible alternatives and explained their advantages as well as their limitations.

Kameleon is a recent tool being actively developed and maintained. Several future functionalities are under development. First, there is a need of recipes for more Linux distributions in order to provide most of the OS used in the High Performance Computing (HPC) domain. The HPC tools and techniques study is a central priority in research platforms dedicated to experimentation. Thus, it is necessary to provide most of the possible systems configurations to experimenters. Other research oriented recipes and steps, dedicated to Resources and Jobs Management Systems benchmarks and tools are also available. Currently, steps to install and automatically configure OAR, Slurm and Torque RJMS are available. These are the basis for the creation of the environment image used to drive our experiments in the next section.

Chapter 3

Workload Traces Replay for Experimental Comparison of RJMS

3.1 Introduction

The work of a Resource and Job Management System (RJMS) is to distribute computing power to user jobs within a parallel computing infrastructure. The RJMS has a strategic position in the whole HPC software stack as it has a constant knowledge of both workload and resources. Its principal goal is to satisfy users demands for computation and achieve both a good performance in overall system's utilization while keeping a good user satisfaction level by efficiently assigning jobs to resources. This assignment involves three principal abstraction layers: the declaration of a job where the demand of resources and job characteristics takes place, the scheduling of the jobs upon the resources and the launching and placement of job instances upon the computation resources along with the job's execution control. Thus the work of a RJMS can be decomposed into three main, interconnected subsystems: Job Management, Scheduling and Resource Management. The latest years each subsystem of the RJMS has become more and more complex.

The advent of multicore architectures along with the evolution of multi-level/multi-topology fast networks has introduced new complexities in the architecture and extra levels of hierarchies that needed to be taken into account by the resource management layer. The continuous demand for computing power by applications along with their parallel intensive nature (MPI, OpenMP, hybrid) made users needs more demanding for robustness and certain quality of services; issues that have to be taken into account by the job management layer.

However, the part of the RJMS which hides a lot of complexity of management of the system is the scheduler. Complexities on resource and job management subsystems may be again encountered in this subsystem. The more advanced features are supported upon the RJMS, the more complex gets the process of scheduling.

Support of features like hierarchical resources, topology aware placement, energy consumption efficiency, quality of services and fairness between users or projects are managed as extra parameters in scheduling and all induce an additional complexity upon the whole process. Furthermore, this complexity will be increasing with the growing size of supercomputers.

Hence with all these new factors to take into account the efficient assignment of large number of resources to an evenly large number of users becomes more complex and in order to deal with the new challenges, experimentation and evaluation of these types of middleware is essential. Inevitably the research of those systems implicate various procedures and internal mechanisms making their behavior complicated and difficult to model and study. In addition, every different mechanism depends on a large number of parameters that may present dependencies amongst them. Thus, it is very important to be able to study and experiment with the RJMS as a whole under real life conditions.

In this work we present a large-scale, reproducible experimental methodology for Resource and Job Management Systems based on the usage of real workloads extracted from production systems. A workload is replayed on several given RJMS configurations to evaluate their performances. An important aspect of the methodology is the use of the workload in the same context as it was in the original system. This context is reconstructed from the observation of the system state at the workload extract start. The methodology allows us to study particular aspects like scheduling without hiding the complexities of the rest of the system. The results can be used, amongst others, as insights to improve the design and internals of the actual RJMS or to modify its configuration in order to obtain better results for the certain workload. In our study we have used two open-source Resource and Job Management Systems: OAR [14] and Slurm [119].

The experiments took place on the Grid'5000 platform [15] and the workload trace was collected from the Curie (BULL/CEA-DAM) cluster which was placed 11th among the 500 most powerful supercomputers on November's 2012 Top500 list[1].

Evaluation Based Upon Real Platform Workload

The first question to ask in the case of real workloads, is which workload log to select and what are the characteristics that we need to study in order to select the right log for the right experiment. For this, an obvious parameter is the size of the system from which the workload was extracted. Indeed the choice of the size of the cluster in the trace file and the size of the one that we will use for experimentation has to be the same and if possible, of the same architecture. Thus, when using a trace for such an evaluation, the total number of resources (cores) has to be the same between the original system and the system where the trace is replayed. The number of cores per node and their hierarchy also has to be respected as this influences the scheduling process. These parameters should be respected in the experimental

[1] http://www.top500.org

evaluation. Several studies were conducted in [54] and in [55] to understand how the nodes topology and architecture influence the global system performance for the same input workload on Slurm RJMS. Depending on the RJMS internal structures, the way how the resources are described to the system may influence the way how they are managed and thus influence the scheduling. However this is not a very strict requirement, the purpose of this constraint is to be as close as possible to the original platform and thus try to mimic the same scheduling scenario. Concerning the size of the platform, i.e. the total number of cores, a study[37] has been driven to propose a method, validated by comparison of performance metrics, to "rescale" workload traces in order to use them on a smaller/bigger platform. Despite the difficulty of the approach described, this is a very interesting topic that should be further studied and validated with the help of workload models.

Another important concern was how we can select specific parts of the workload logs in order to use them on our experiments. In our experimentation methodology we are limited by the time, because a normal single experiment execution should not last more than few hours. However, the logs maintained in Parallel Workloads Archive [1] depict usually the utilization of the system for at least 4 months. Hence we need to define criteria so as to guide our selection of particular parts of the workload logs. Which parts contain more valuable information for replaying? Which parts will provide more interesting behavior for observation?

As discussed on [48], underloaded systems present typically similar performances[2], whereas higher load conditions stress the system and expose differences in how systems react to load. In a previous study [58] with real workload deployment one of the first adopted characteristics that guided our selection, was the system load. A specific toolkit has been developed to calculate the system utilization of a workload log for specified duration window. Therefore, various workload parts with different system loads were used to test the system under different levels of stress. A similar approach has been used in this study. In particular, as we will see in the next section, there was a more detailed selection observing the jobs interarrival and waiting times through manual intervention.

One of the most known workload formats which is widely used in this type of studies is the standard workload format [16] (SWF) described in Section 1.1.

3.2 Evaluation Process

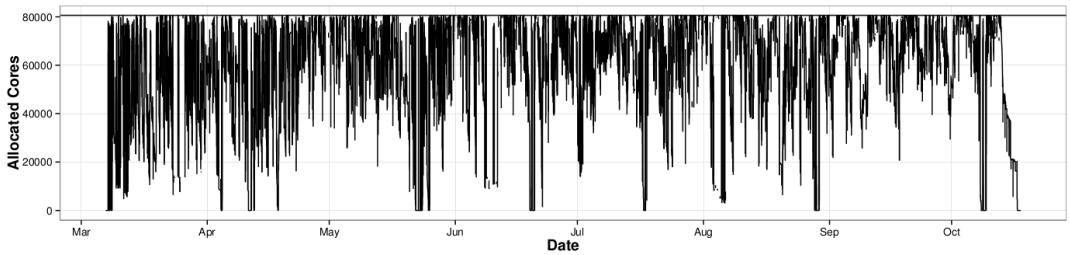
Commonly the works on scheduling and RJMS tool evaluation use models to assess the performance of a given algorithm, RJMS or system configuration. Until recently, these models were run in a simulated environment to abstract the complexity of setting up a full large scale distributed infrastructure. In [54], then in [55], the

[2]it is true if the performance metric is one of the commons: system utilization, wait time, slowdown, throughput, however if the performance metric is the energy used by the platform, an underloaded workload may help evaluating several energy saving techniques and highlights their differences.

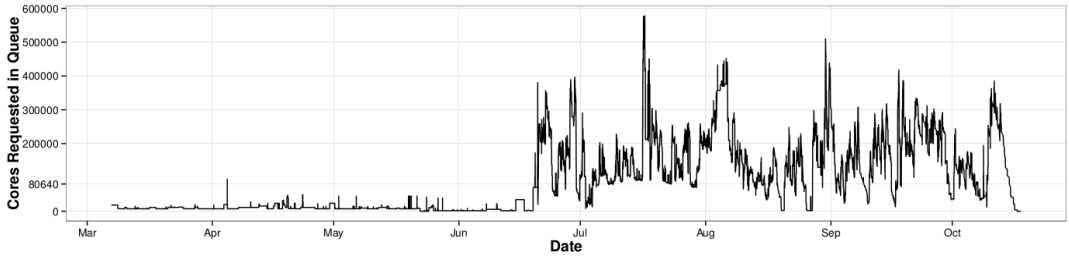
evaluation of RJMS has been driven by replaying the ESP [118] synthetic workload or modified versions of it, upon the large scale experimental platform Grid'5000 [15] and other large platforms. Driving the experiments at large scale rather than in simulation enabled to have a more accurate view of what would be the system behavior in a real platform. Sophisticated models that can be used for RJMS evaluation are workload models based on the statistical study of real logs from production systems. Thus when using a model or synthetic workload for evaluation, the quality of the whole evaluation process is based on the good choice of the model and its parameters. In [30], the authors compare the benefits and drawbacks of using either a real workload or a model. The choice of using a model to generate a synthetic workload in order to evaluate a given system is interesting because a model by definition describes a general behavior of the workload and it is easier to use than a real trace extract. However an important drawback of using a model to fit a given observed workload, as for a trace extract is the representativeness. Using a model to generate a workload similar to one other observed workload is a complex task. A model sophisticated enough requires that its parameters are well adapted to the kind of workload we want to generate to provide a good synthetic workload. This requires both a good knowledge of the workload to fit and a deep study of the correlations between the events to justify the choices of the parameters. Such models were derived from the observation and statistical study of real workloads to extract their interesting features. The most recent models for generating cluster workloads are quite old [20](year 2001) and [82](year 1999) and we cannot be guaranteed that the observations that led to their construction are still valid as the platform sizes have evolved rapidly and users computing power needs have increased consequently. This increases the risk of a resulting workload not being representative as nowadays systems have to deal with hundreds of thousands of resources to manage. Further research on this topic should be conducted to analyze what changed in the users and systems behaviors between nowadays machines and 15 years ago. In this work we propose instead of using a workload model, to use a real trace extract from such a recent system and to replay it at large scale. This means that the trace extract will be replayed in an environment comparable to its original environment. In this case, the use of the trace extract will leverage the full complexity of the workload to evaluate a given RJMS configuration for a given scenario: the trace extract chosen. If chosen carefully, the trace extract will contain all the richness of the original workload without lacking representativeness. A trace extract is a scenario: it can be representative of an average behavior or it can be representative of a very particular pattern as for high throughput peaks mixed with low average utilization, cycles in the arrivals patterns, or even overload of the system. Thus a trace will be representative of what it was selected for.

Getting the Trace

The evaluation of RJMS with workload traces can be driven with traces that come from real platform logs or synthetic traces built from models. In this work we propose to use a trace from a recent large scale production platform. This trace



(a) Utilization of the SWF Trace on Curie.



(b) Queue Size (total cores requested) of the SWF Trace on Curie.

Figure 3.1: Utilization (3.1a) and Queue Size (3.1b) of Curie during the trace reference period.

comes from the **Curie** SuperComputer[3], operated by the French Atomic Energy Research Center CEA[4]. Curie is the first French Tier-0 system open to scientists through the participation into the PRACE[5] research infrastructure. Since its upgrade in February 2012, Curie Thin-Nodes consists of 280 bullx B chassis housing 5,040 bullx B510 nodes, each with two 8-core Intel Sandy Bridge processors for a total of 80640 cores. The conversion of the logs and the obfuscation of sensitive data was done in cooperation with CEA operators. For an easier use and redistribution, Curie logs have been converted to the Standard Workload Format (SWF) described in Section 1.1.

We extracted and converted to SWF a trace taken from the period after the upgrade where the machine was at its full capacity. This leads to a trace from March 2012 to Mid-October 2012 (date of the SWF trace build). From this we removed administrators jobs as these kind of jobs are only for testing purpose and do not take part of the production workload.

To give an insight of Curie usage, Figure 3.1 presents the total core allocation (utilization) by the RJMS and the queue size (sum of all the jobs' core requests in the queue) for the Curie trace. It is remarkable that after June 2012, the size of the queue is very high (several times the full platform capacity). It is also noteworthy

[3]http://www-hpc.cea.fr/en/complexe/tgcc-curie.htm
[4]http://www.cea.fr/english-portal
[5]http://www.prace-project.eu/

that big peaks in the queue tend to correspond to very low or zero utilization, these peaks correspond to maintenance periods or system downtimes. In this extract of about seven months, more than 87% of the jobs have a short runtime (less than one hour) and almost 82% have a very short runtime (less than 20 minutes). However the distribution of the time requests by the users reveals that 75% of the jobs have a request of more than two hours. This indicates that the impact on the scheduling decisions is quite strong and that backfilling policies are harder to adapt. The distribution of the cores requests by the users shows that users generally request few cores for their jobs regarding the platform capacity. 30% of the jobs request one node or less, the median cores request is 32 cores (2 nodes) and the five last deciles are respectively 32, 128, 256, 512 cores and almost the full platform for the last decile. In this trace from March to October, the inter-arrival time distribution reveals that 30% of the total amount of the jobs in this period were submitted less than 1 second after their predecessor submission, which means that many jobs are submitted at the same time. The median inter-arrival time between two jobs is 3 seconds and the 87th centile is one minute. We observe that most of the jobs (87%) tend to arrive rapidly within the minute after the previous job submission. This compute is done without considering the daily cycle but as by night few jobs are submitted in this cluster, it is a good approximation of the throughput the system has to manage during working hours.

Criteria for the Selection of the Trace Extract

To select a trace extract from the whole workload trace we focused on several criteria. First, the period of the extract should show a high utilization in terms of allocated cores and a high number of cores requested in queue. This ensures that the extract will not belong to a period with low activity. As in the following we will drive experiments on the ability of the RJMS to process a high job load we ensure that the selected trace extract will match this criterion. If the purpose of our experiments would have been testing energy saving or co-scheduling techniques, a different kind of criteria would have been chosen, e.g. an average low utilization with some short high utilization peaks.

The duration of the extract to replay should be short; the order of the hour is acceptable and the inter-arrival time should be quite small as we have to replay the trace extract respecting the arrival times of the jobs. This is necessary to guarantee that the submission activity is high and that the replay experiment will not take too long.

Replay of a Trace Extract

In order to replay the extracted trace we developed the **"riplay"** dedicated tool, written in Ruby[6]. It takes as input an SWF trace, extracts the useful information and

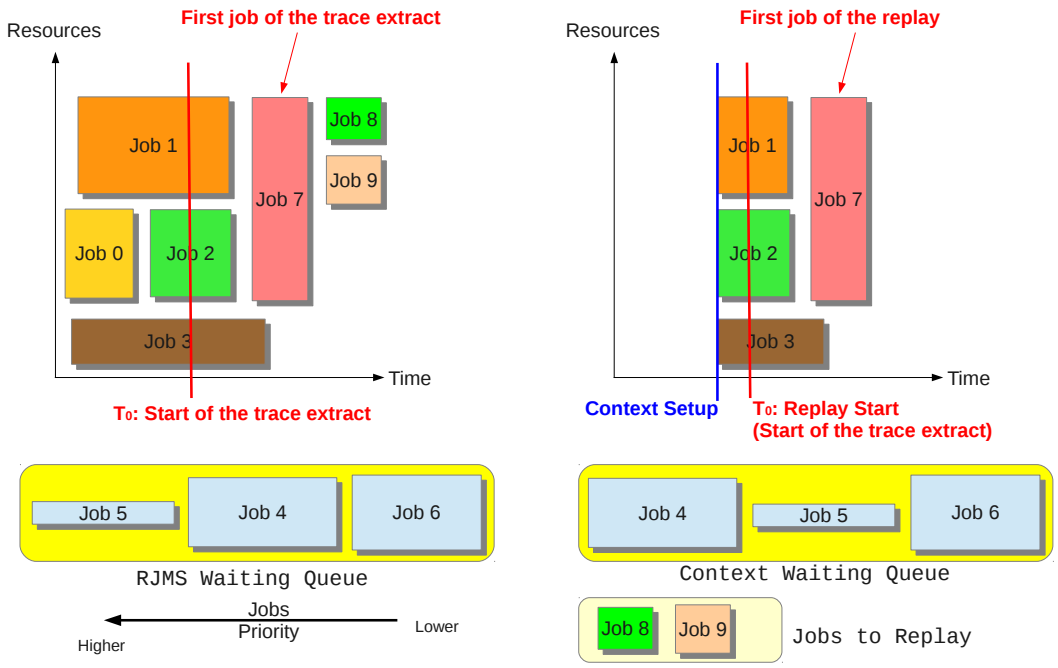
[6]for about 2000 Lines of Ruby Code

submits the sequence of jobs from the trace to a given RJMS. The order of arrival of the jobs, their inter-arrival times, and resource and time requests are respected during the replay. As the goal of this tool is to test how the RJMS reacts to a given trace, the result of the jobs themselves is useless. Thus, replayed jobs do not do anything and are just composed of *"sleep"* commands that wait for the original job duration. With this we ensure that replayed jobs will run for the same duration than in the original log. The tool first loads the SWF file that represent the extract of the trace to replay, builds the list of jobs to replay in a structure containing all the submission information and arrival dates. When the replay starts, for each arrival date the tool will submit the jobs to the RJMS. The submission is done by wrapping the job submission parameters to the format of the RJMS on which will be done the replay. Currently two RJMS are supported: OAR and Slurm. The goal of the "riplay" tool is to guarantee that the jobs are launched as they were submitted by real users and that inter-arrival times and important jobs characteristics (runtime, requested time, number of cores) are respected.

Trace Context

When we replay a trace extract we have to understand that this extract is totally out of its original context. During the trace extract selection process, once the interesting section has been marked out beginning from a given job (or timestamp) to an other one, only the jobs submitted during this interval will be selected as belonging to the extract. However this is wrong in the way that at the submission time of the first job of the trace, the original system was probably not empty, other jobs might be running or in the queue. If we replay the raw trace extract this means that at the start of the replay the system will, this time be empty and the replay would certainly be very different from what happened in the original situation. To counter this bias we propose to include in the replay (actually just before the replay) a set up of the original RJMS state. Thus, we capture all the jobs that were in the system (running or queued) at the trace start and reset them in the same state before the actual replay. We have thus three trace extracts corresponding to three sets of jobs: the extract of the jobs to replay, the one of the jobs that were queued and the one of the jobs that were running. We improved "riplay" to take into account this context setup.
Now, when a trace replay is launched, the first step is the submission of all the jobs that were running. To ensure that they will end their execution in the replay at the same time they had in the original workload their time characteristics are modified. Their runtime and requested time are reduced of the duration they had stayed in the running state at the start of the trace extract in the original workload run. Then jobs that were queued are submitted. To ensure that the jobs form the "running" set will all start before the "queued" set and before the replay, we submit all the jobs from the running and queued sets as "Hold". This special mode of submission, supported by most RJMS allow the user to submit a job that will not be scheduled until it is released explicitly by the user. With this feature we can submit sets of jobs that will not be scheduled immediately and then control their management by

the system when necessary. This method will speed up the submission process and ensure that all the jobs from the "running" set will start at the same time and when they are actually all in the running state we release the jobs from the "queued" set and start the replay immediately. With this mean we ensure to guarantee the order of submission and the set up of the environment as in the original workload.



(a) Trace Context Construction. (b) Trace Replay.

Figure 3.2: Trace Context Construction from the Original Trace and Trace Context Reconstruction During the Replay.

Actually, the idea of the context reconstruction looks like the idea of the snapshot. We determine a period in the original trace and mark it as our trace extract. At the start of this extract we do a snapshot of the status of the system (running jobs, queued jobs, jobs to come). Then, from this snapshot we can replay the same trace over and over according to different configuration to see how they behave. This enables to do as if we were able to pause the system, change the configuration and resume, then see what happens. This as much as the experimenter wants.

Figure 3.2 illustrates the process of context reconstruction and trace replay based on the original trace observation. In Figure 3.2a we first determine the starting date (T_0) of the trace extract, which is represented in the figure by the vertical red line. All the jobs that were terminated before T_0 will not be part of the replay. It is the case for Job0. At T_0, Job 1, 2 and 3 are running, thus they will be set as running during the context reconstruction (see Figure 3.2b). Note that the ending date of

these jobs will be the same in the original trace (3.2a) and in the replay (3.2b). At T_0, Job 4, 5 and 6 are in the queue. In the original trace, the RJMS policy decided to set Job5 priority higher than Job4. For the replay, these 3 jobs will be set in the queue but as the replay RJMS policy might be different from the one of the trace, we reset the priorities and let the RJMS re-assign the priorities itself during the context reconstruction period. At T_0 the trace extract starts and the jobs 7, 8 and 9 will be marked as jobs to replay. They will arrive according to their submission date after T_0 and in the same order than in the original trace.

Warm up Period vs Steady State

As we reconstruct the original submission context before the replay we introduce potentially a high and artificial load on the RJMS during the context set up. A lot of jobs have to be submitted and scheduled. The impact on the RJMS can be quite strong and when we analyze the results of the scheduling metrics of the replay we need to not take into account this environment setup as it may overshadow the replay itself. We will thus distinguish 2 periods: the warm up period corresponding to the environment set up and the steady state that starts at the time the first job of the replay is submitted. This also enables to really take into account in the evaluation only the jobs that belong to the replay as we do the evaluation on the steady state only.

Limitations of Replay: Comparing a Workload Log and a Replay

The replay of a trace extract is however limited in itself. Although it is possible to compare two replays it is invalid to compare a replay with what was originally observed in the workload. Other characteristics like Quality of Services (QoS), Fairsharing or machine failures cannot be easily reproduced. QoS and Fairsharing policies change the scheduling priorities and it is not always possible to retrieve what was the Fairsharing score for a given user at a given date. Machine failures also impact the utilization of the platform and even though this information sometimes can be retrieved, the SWF format in itself does not contain such information. A machine failure model can also be used, based on the study of computing machines failures [96, 74]. In spite of this constraint it is however possible to compare replays done with the "riplay" tool as long as they have the same parameters and experiment conditions. We validated the reproducibility of the replay by comparison of the scheduling decisions made with the same configuration for different replays. The results were every time the same. The method of the replay is thus an interesting way to compare different RJMS or RJMS configurations with a given workload trace extract. As the context of replay is the same and the sequence and characteristics of jobs is respected, the replays highlight the scheduling choices of the different configurations and thus enable their comparison.

3.3 Experiments

Experiments Environment

Experimental Testbed

All the experiments have been driven on Grid'5000[15]. Grid'5000 is an experimental platform intended to enable researchers to experiment at large scale by providing a set of tools and the infrastructure itself. On Grid'5000, users can deploy their own customized operating system and full software environment on the nodes and have full control of their experiment.

The Grid'5000 images used to run our experiments are built with Kameleon; this ensure the perennial reconstructability and diffusion of the experiment environment and results reproducibility.

With this experimental infrastructure we were able to run one given experiment set several times (around ten) to guarantee the statistical value and validity of our results; the results were identical for every run.

Use of Emulation

For replaying the extracted trace, we choose to use an emulation method on the computing nodes. Both OAR and Slurm enable to use the same physical node as several emulated computing nodes. This choice is justified by the fact that as the jobs do not really compute (no code is actually run by the jobs as the goal of the replay is to analyze the RJMS response and not the jobs') it is useless to use a real scale experiment and worst it would correspond to a big loss of compute power and energy to replay at real scale. Although we use emulation, the hierarchy of the computing resources is respected in our experiments[7] and the emulated infrastructure is the same than the original one apart form the fact that each physical node represents several computing nodes. Thus emulation enables to do the experiment without the need of an infrastructure of the size of the original one. This has the great benefit of not spoiling the computing resources of the platform for evaluation and to keep it in production mode. The purpose of using emulation is only for enabling the large scale experimentation on a limited amount of physical nodes. The evaluation methodology proposed is valid for both emulated or real scale experiments.

Trace Chosen for Experiments

The Curie trace extract chosen to run the experiments was a one hour trace, taken from the Monday 1st of October 2012 from 11:00 a.m. to 12:00 noon. This trace extract is interesting to replay because it fits our extract selection requirements described in Section 3.2. It is not too long and has a high utilization with a queue size very high that increases during the selected period. Figure 3.3 and Figure 3.4

[7] see Section 3.1

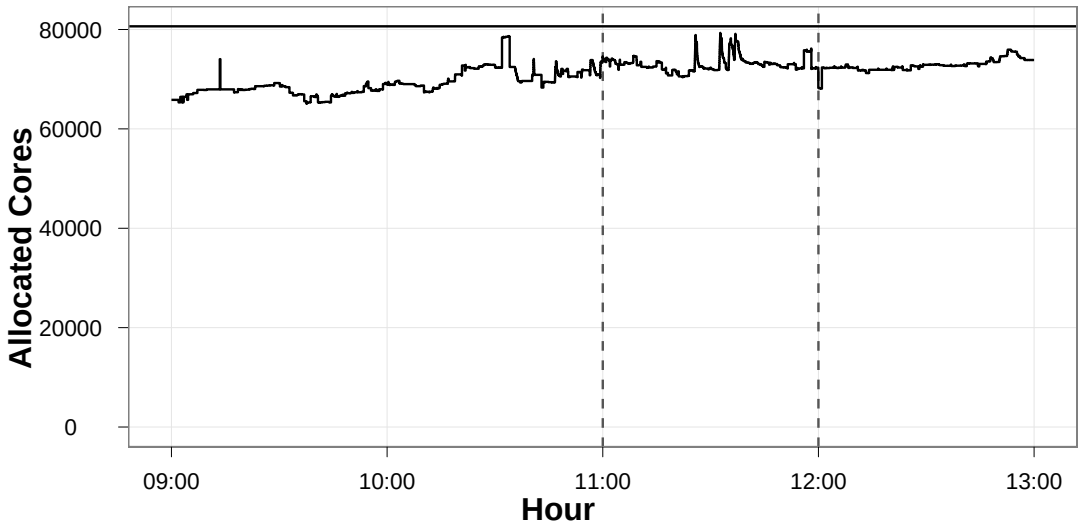


Figure 3.3: Utilization on Curie around the selected trace. The dotted vertical lines delimit the selected trace. No comparison can be made with replayed traces, see paragraph "Limitations of Replay" in Section 3.2.

show respectively the utilization and the queue size on Curie of the 1st of October morning. Vertical lines delimit the selected trace.

As explained in paragraph "Limitations of Replay" in Section 3.2, no performance comparison can be made between the original trace and the replay. As we do not have all the information about QoS policies and users Fairsharing scores, the scheduling between the original trace and the replay might be different, as for the ordering of the priorities of the jobs, leading to differences. These figures are presented only to give an insight of the state of system state at this time. It is what happens in Figure 3.3 where several utilization peaks are caused by users with a higher priority that come in the system and submit jobs that are scheduled to start shortly. In the replay, as the priority score between users is not the same as in the log, the observed result is different. Other kinds of differences like this one can be observed between a workload log and its replay.

Before the extracted trace starts, the load on the cluster and the submission rate increased progressively from 9 a.m. time where the users submission started to increase substantially on the cluster. At 11 a.m. the utilization was very high with a high queue size and short jobs interarrivals. In the selected extract most of run times are short, on the order of a few minutes. Their distribution is approximately the same as in the full month of October. On the extract the inter-arrival time is shorter than during the month of October. This more dense submission pattern is interesting because it presents a higher throughput and thus a higher system stress.

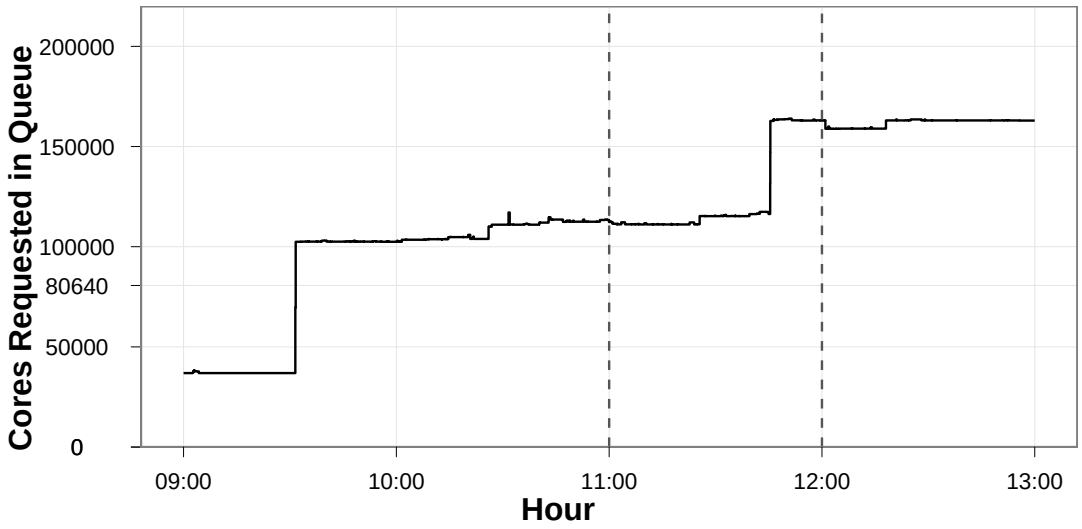


Figure 3.4: Queue on Curie around the selected trace. The dotted vertical lines delimit the selected trace. No comparison can be made with replayed traces, see paragraph "Limitations of Replay" in Section 3.2.

Evaluation of OAR's Scalability

OAR

OAR[14][8] is a versatile RJMS that aims mid-sized computing infrastructures (on the order of the thousands of resources). It is currently used in both production (as the CIMENT[9] mid-sized computing platform) and research (like Grid'5000) infrastructures.

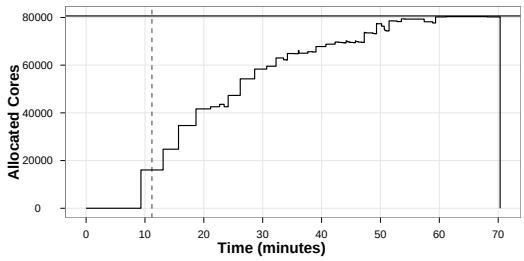
OAR is provided with a FIFO conservative backfilling scheduler that supports fair-sharing. This fair-sharing feature was not used in our experiments as we could not have the original users fair-sharing scores from the original trace. An other scheduler called Kamelot, not packaged by default with OAR is also available. It is designed to be more scalable than the packaged one, although less versatile. The Kamelot scheduler is also based on FIFO with conservative backfilling.

In this section we replay the selected extract on a reconstruction of the Curie infrastructure on Grid'5000 (same total number of resources, same resources hierarchy[10]). Then we compare the results of the replays on the two different schedulers.

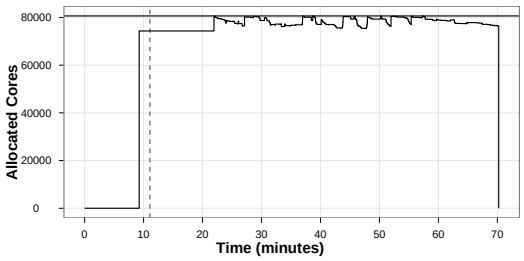
Replay Curie Traces on OAR

Figure 3.5a shows the utilization and Figure 3.5c the queue size of the system during the replay of the trace extract from Curie. The vertical line at the 11th minute

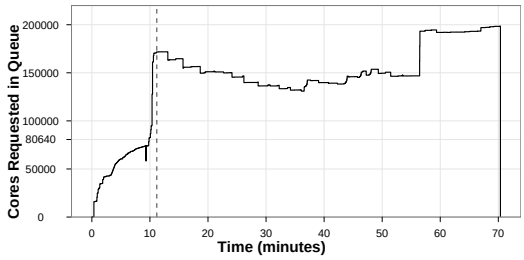
[8] http://oar.imag.fr
[9] https://ciment.ujf-grenoble.fr
[10] see Section 3.1



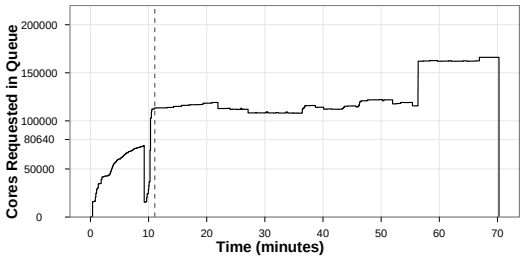
(a) Utilization with the packaged scheduler.



(b) Utilization with the kamelot scheduler.



(c) Queue with the packaged scheduler.



(d) Queue with the kamelot scheduler.

Figure 3.5: OAR utilization and queue size for the 2 schedulers replaying the Curie selected trace.

shows when the warm-up phase stops. From minute 0 to minute 9, running jobs are submitted (note that it takes 9 minutes to submit 130 jobs sequentially to the system). From minute 9 to minute 11, queued jobs are submitted. Then the steady state begins.

On Figure 3.5a, we observe that at the end of the warm-up state, the context is not set up. It takes too much time for OAR with the default packaged scheduler to schedule on such a large number of resources. As the warm-up process takes too much time, it could not guarantee the constraint of the jobs states (running or queued) at the replay start (steady state). Thus the replay result is not valid and we will not consider further this experiment. In the case of the default scheduler provided with OAR, the system cannot scale up to the Curie trace extract. Logs show that this slowness is due to a too high scheduling computation time.

Kamelot, Large Scale Scheduler for OAR

Driven by the scalability problems of the default packaged scheduler, identified by our evaluations, a new scheduler has been developed. The idea of this scheduler is to redefine the way how resources are stored in OAR's internal data structures, enabling the scheduling to be faster and the resource matching to be very efficient. In this section we apply our methodology on this scheduler.

With this scheduler, the warm-up process behaves normally and at the end of the warm-up the system is in the reference initial state enabling to replay the trace extract.

The utilization is on average 96% during the steady state. The whole utilization of the system (Figure 3.5b) reveals an interesting behavior: from minute 11 to minute 22 the utilization does not change at all. At minute 10, 26 jobs are submitted to the system (see figure 3.5d to see the increasing number of cores requested in queue). At this time, OAR logs recorded a scheduling computation that last about 11 minutes. This shows that the Kamelot scheduler has improved OAR's scalability, but further work needs to be done to improve the throughput.

The replay of the trace extract with its context in OAR with Kamelot scheduler revealed that OAR is now able to process a trace from a PetaFlopic cluster such as Curie but still has throughput problems (in terms of jobs submission).

Slurm, Should we Favor Small Jobs?

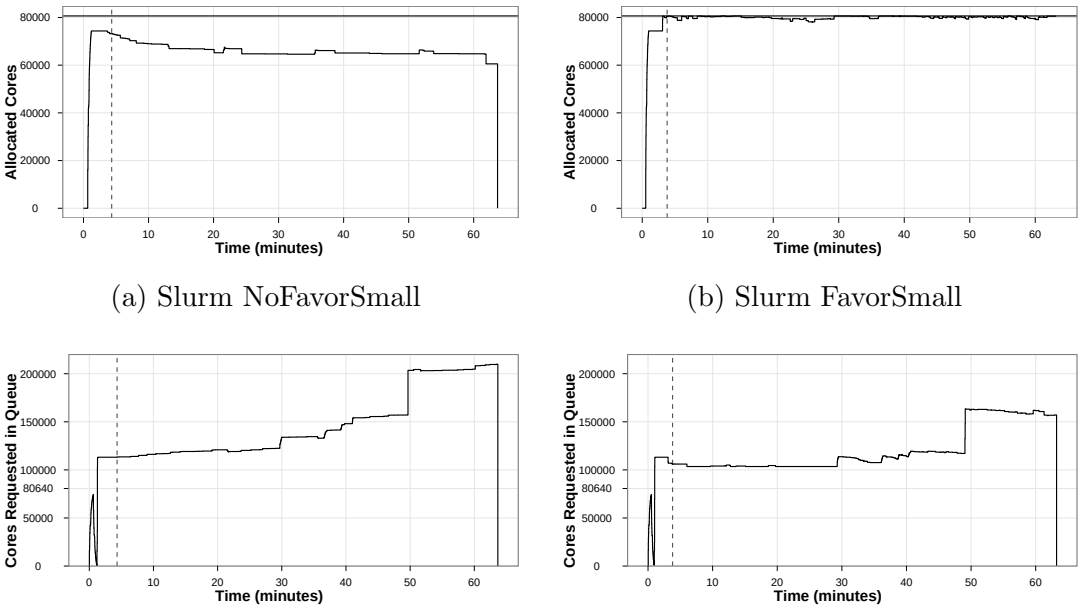
Slurm[11] is a highly scalable RJMS developed for all kind of cluster. It's actually used in best Top500 machines like Stampede[12], Tianhe-1A[13] or Curie.

Slurm's scheduler is a FIFO scheduler with conservative backfilling and many fair-sharing and Quality of Services (QoS) options. In the following experiments, configurations used are the same than in Curie but with some features disabled like fair-sharing, QoS or topology-aware algorithms.

As seen previously, Curie workload is composed of a lot of small jobs (in terms of number of cores). In this section we study the effect of a configuration parameter of the scheduler: *PriorityFavorSmall*. This parameter indicates if the scheduler should favor or not small jobs, i.e. give a higher backfilling priority to the jobs in the queue that request the less number of cores. With *PriorityFavorSmall* activated (option set to true in the configuration file), the smaller the job is, the smaller its size factor will be. The job's size factor is then computed with the job's other characteristics to give a score for the priority queue[14]. In the following experiments two configurations are studied; they are the same apart from the *PriorityFavorSmall* feature which is enabled or not. The configuration with this feature enabled is called *FavorSmall*, and the other *NoFavorSmall*.

Figure 3.6 presents the replay results for the two configurations *NoFavorSmall* and *FavorSmall*. For both configurations, the warm-up phase is faster than for OAR, it takes only 4 minutes. This comes from the fact that Slurm is faster to process a job submission than OAR. Figure 3.6a shows a utilization that is not very high. Table 3.1 shows that few jobs were launched during the replay while the queue contained a lot of jobs. It has been previously measured that in the Curie workload most users estimate badly their execution time. As in *NoFavorSmall* bigger jobs have the biggest priority, Slurm is making some room for them by not launching further jobs. Thus small jobs cannot be backfilled in the available space because they requested too

[11] http://slurm.schedmd.com/
[12] http://www.tacc.utexas.edu/resources/hpc/stampede
[13] http://www.nscc-tj.gov.cn/en/
[14] https://computing.llnl.gov/linux/slurm/priority_multifactor.html



(a) Slurm NoFavorSmall (b) Slurm FavorSmall

(c) Slurm NoFavorSmall (d) Slurm FavorSmall

Figure 3.6: Slurm utilization and queue size for the 2 configurations replaying the Curie selected trace.

much time. This phenomenon results in the observed under utilization of the cluster. In *FavorSmall* the situation is different. Small jobs are launched first.

As their runtime is far shorter than their requested time, the system is finally able to launch more jobs. This results in an very high utilization of the cluster.

Other usual measures, like wait time or stretch time are not comparable due to the low number of jobs launched in the *NoFavorSmall* configuration.

This study has shown that in the selected trace, *PriorityFavorSmall* impact is mostly due to badly estimated run times. To improve performance while having a higher priority for bigger jobs, it would be interesting to have a scheduler that takes into account badly estimate run times.

OAR Kamelot vs Slurm FavorSmall

In this section, only steady state is taken into account. This to ensure that we really compare only the replay part of the experiment. Table 3.1 presents the results of our different experiment runs regarding their configuration. As the results were identical for each experiment set, the standard deviation is 0 for each metric. As seen previously, Slurm FavorSmall seems to have the best result for this trace extract. For the same workload, the average utilization has been improved by 17% using this option. OAR Kamelot shows also interesting results. In the default scheduler it is noteworthy that OAR does not scale. With Kamelot the average utilization is

Configuration	Jobs Completed	Jobs Launched	Average Utilization
OAR default	8%	28%	77%
OAR kamelot	58%	89%	96%
S. FavorSmall	63%	95%	99%
S. NoFavorSmall	12%	30%	82%

Table 3.1: Utilization, jobs launched and jobs completed for different configurations.

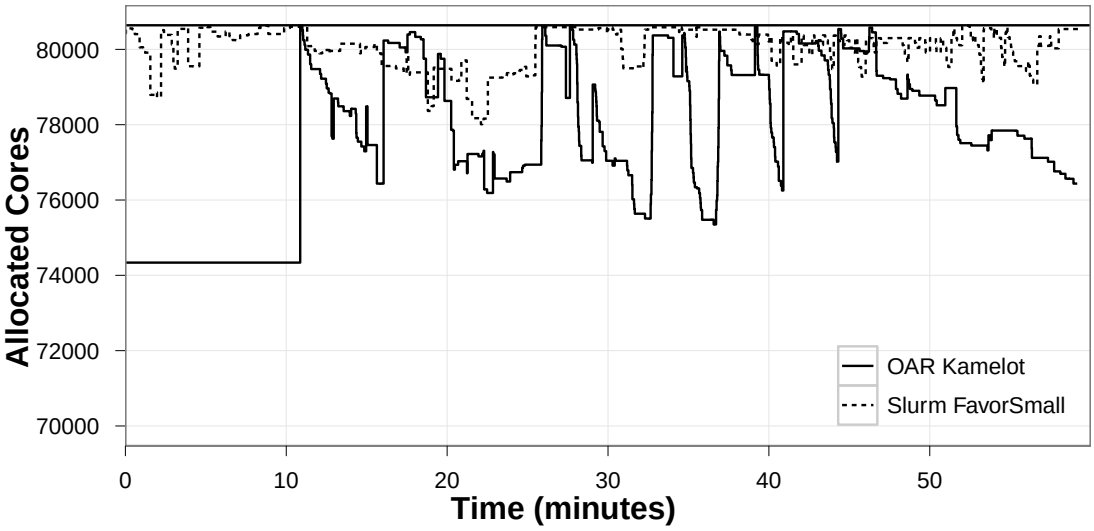


Figure 3.7: System Utilization for OAR Kamelot and Slurm FavorSmall Configurations

increased by 19% and the scalability is acceptable. The two other configurations show bad results in terms of number of jobs launched.

Figure 3.7 presents a zoom of the System Utilization for both OAR Kamelot and Slurm FavorSmall. We observe that Slurm has still a better utilization than OAR. When comparing the values from Table 3.1 it is noteworthy that Slurm FavorSmall launched more jobs than OAR Kamelot and more of these completed during the 1 hour experiment. With a higher number of jobs launched and a better utilization, Slurm is better than OAR for system utilization for these configurations on the replayed trace extract.

Figure 3.8 compares CDF of the wait times of the jobs for OAR and Slurm. Not 100% of jobs are represented on this graph because jobs that are not launched have a infinite wait time. In this figure we observe that jobs wait generally less in OAR than in Slurm. Half of the jobs wait less than 2 minutes in OAR, whereas the median wait time is 3 minutes for Slurm. However it is noteworthy that OAR was able to run less jobs; it can be observed that for a wait time of 14 minutes, the two CDF curves crosses. This gives the idea that Slurm is performing better in the long term with

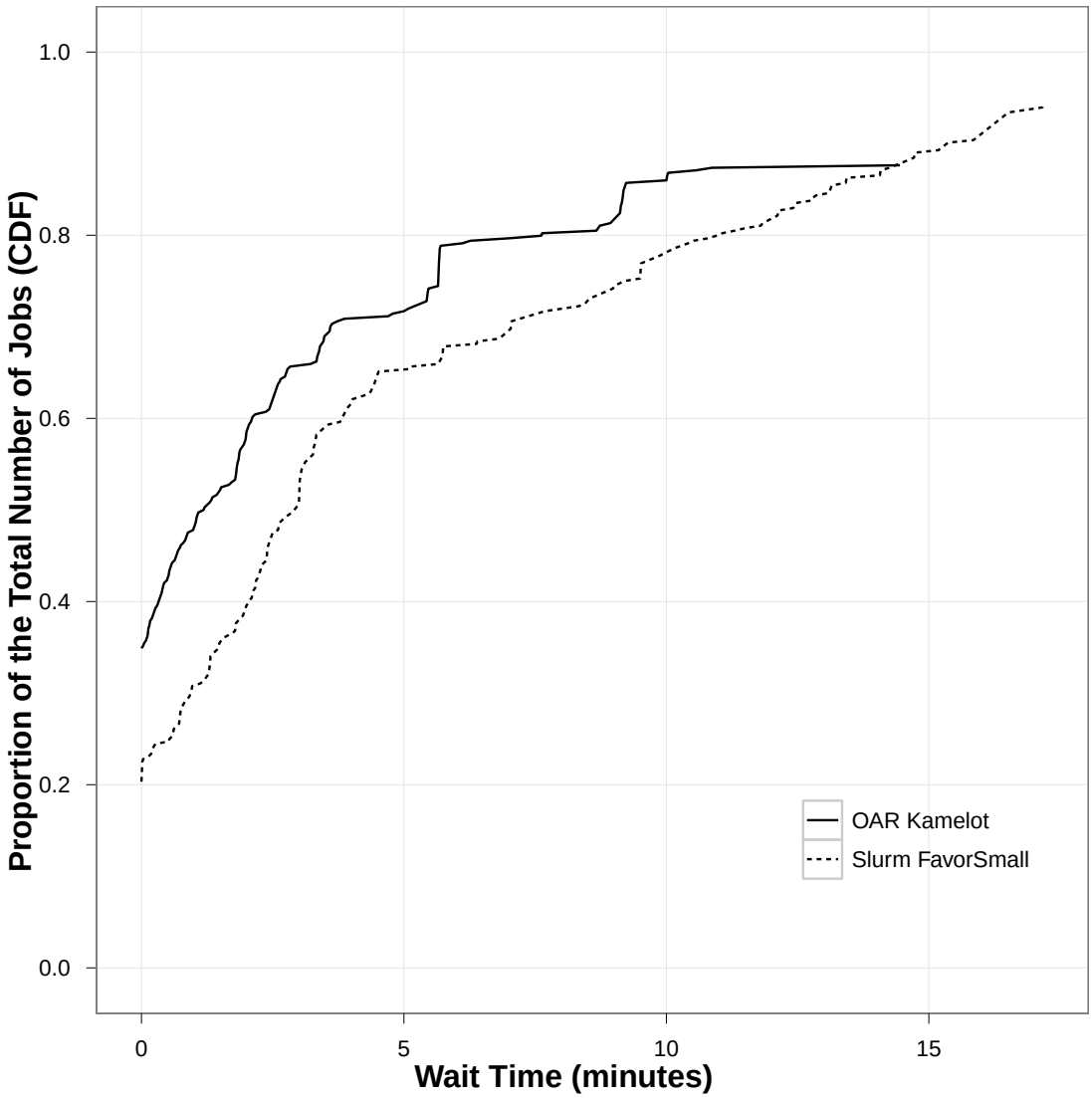


Figure 3.8: CDF of Wait time for OAR Kamelot and Slurm FavorSmall jobs. Jobs that are not launched have an infinite wait time.

more jobs.

Figure 3.9 presents the CDF of the Slowdown (stretch factor) for both configurations. Here, only jobs that are launched and ended have a finite stretch factor. The Slowdown represents the ratio of the time spent by the job in the system (time queued + run time) over the run time of the job. For this metric, the closer to 1 is the value, the best it is for the job. A Slowdown of 1 means that the job did not wait, a Slowdown of 2 means that the job waited as long as it ran.

The Slowdown shows that OAR performs way better in terms of fairness than Slurm on this trace extract. For Slurm, 14 jobs had a very high Slowdown (between 30 and 50) and were not presented in Figure 3.9. For this experiment OAR was more fair

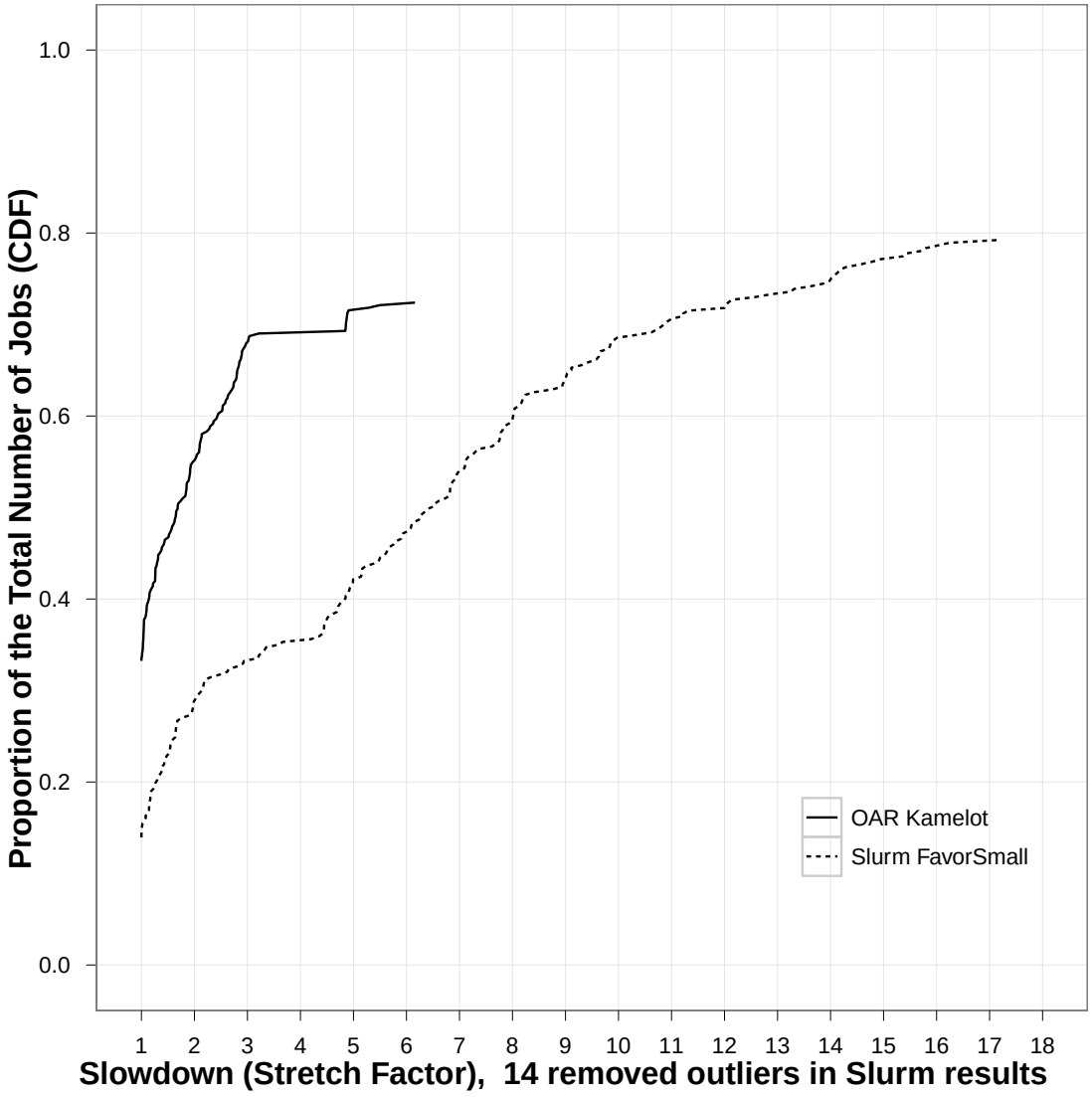


Figure 3.9: CDF of Slowdown for OAR Kamelot and Slurm FavorSmall. Jobs that are not ended have an infinite slowdown.

in its allocations and this resulted in a low Slowdown (maximum is 6). Nevertheless, the results here do not take into account fairsharing which is an important parameter in scheduling and would change the results. Testing fairsharing feature in this type of experiment that tries to mimic the original workload environment demands the reconstruction of fairsharing scores from the workload. This is a rather difficult process that we will be focusing in the future. To be able to do this, we need an history of the users fairsharing scores. However, if the purpose of the experiment is not to reproduce as strictly as possible the original platform, it would be an easy and interesting experiment if we attribute empirically fairsharing scores to users before running the replay, to see the impact of these for a given workload.

In conclusion for this trace extract, using FavorSmall on Slurm enabled a higher System Utilization and more jobs launched and completed than OAR Kamelot. However, this feature has the drawback of creating an unfairness between the jobs regarding the time spent in the queue. For this experiment, OAR despite a lower number of jobs launched showed a very good Slowdown and thus a very good fairness in the wait times regarding the jobs length. The problem of increasing the slowdown of the jobs with a higher platform utilization is not new [93]. However in this case for the difference of utilization between OAR and Slurm, the slowdown increase is much higher. A 3% system utilization from OAR to Slurm involved a slowdown that was more than doubled. This seems to come from the fact that with this FavorSmall policy, Slurm generates a sort of starvation for the large jobs as long as there are small jobs. Despite a better utilization, using this policy is not acceptable in a user fairness point of view. However the idea is interesting at may be followed to find a better compromise. The small jobs priority is probably set way too high and with a better tuning this would result in an interesting trade-of. This kind of "real life" experimentation approach is a convenient way to deeper study the configurations and tuning technique changes and see their impact in a realistic and experimental way.

Further experiments have to be driven to compare throughput as it seems to be a weakness in OAR and it would be also interesting to evaluate the scalability of Slurm and OAR by adding the fairsharing parameter. Finally various workload extracts should be used to obtain an in-depth evaluation under different types of workload.

3.4 Conclusion & Future Work

In this work we propose a new experimental methodology for RJMS performance evaluation. This methodology is based on the replay of a workload trace extract at large scale with reconstruction of the context. We illustrate this approach by the comparison of two scheduling techniques on two different RJMS on a selected trace extract. The specific workload used in our study comes from a production platform of a PetaFlopic supercomputer. Reconstructing the replay context enables to start the replay of the trace exactly in the same state as the original workload. This eliminates the bias of including the warm-up period during a trace replay as we start the replay directly on the steady state.

In our testbed we used an emulation technique provided by OAR and Slurm which allowed us to get a large-scale experimentation with small scale deployment. Furthermore, the analysis of the global one year workload enabled us to select a one hour part that highlights a characteristic behavior with a higher load than average. Hence, the duration of the experimental process is short enough to be run several times.

The methodology formalizes and uses metrics such as system utilization and slowdown, enabling us to judge the efficiency of the scheduling algorithms. It can be used to acquire insights to better study the RJMS under certain workloads and enable its improvements. It led us to several improvements on the scalability of OAR and

possible optimizations for the configuration of Slurm upon large systems with similar workloads. The methodology can be adopted as a tool to help us on the development of new scheduling algorithms which are needed in the road towards the Exascale.

Further work needs to be done to improve the "riplay" tool. Fairsharing has not been taken into account in replays as the users fairsharing scores were not available. This is one of our new directions. Another important aspect would be to take into account hardware and machine failures during the workload extraction then in the replay. This kind of information not necessarily being available, an other option would be the integration of failure models [96, 74] to the "riplay" tool.

Chapter 4

Discussion on Experimental Evaluation

Among the different methods for distributed systems evaluation, experimentation is the closest to reality. By replaying real workloads upon a real infrastructure the experimenter faces up the full complexity of such systems. Workload models, even sophisticated enough cannot contain all the richness of a real workload.

4.1 On the Importance of Context Reconstruction for Trace Replay

When replaying a workload model, the initial state of the replay is an empty queue with no running jobs on the cluster. This is not realistic in the way that this situation happens only a few times in the cluster life. A normal production cluster constantly sees users submitting jobs. Even during maintenance periods or system downtimes, the submission queue is not empty and thus when production restarts the workload submitted by the users is still co-existing with a context.

Actually for the full Curie trace, this happened only 22 times. All of these events were before May 24th 2012, i.e. before the high utilization period of the cluster that started at the upgrade. During the period before the upgrade, Curie was in pre-production mode. Thus we cannot say that in Curie production's period such event happened.

When we compare the results of scheduling efficiency metrics on the same workload with and without taking into account the context, we observe very different results. Table 4.1 presents the comparison of the replay results with and without the context reconstruction for OAR Kamelot and Slurm FavorSmall.

For OAR, without context setup, the maximum wait time for a job is 24 seconds (with a median of 1 second and a third quartile being 7.25 seconds). The maximum slowdown value is 1.44 (with a median of 1 and third quartile being 1.13). For the same configuration, reconstructing the context led to very different results; maximum wait time was about 14 minutes and maximum slowdown was 6, the CDF of wait time

Configuration	Jobs Completed	Jobs Launched	Average Utilization
OAR kamelot	58%	89%	96%
OAR kamelot without context	81%	99.5%	25%
S. FavorSmall	63%	95%	99%
S. FavorSmall without context	81%	100%	25%

Table 4.1: Utilization, jobs launched and jobs completed for the two best configurations with and without workload context reconstruction.

and slowdown for this experiment are available in previous chapter, see Figures 3.8 and 3.9. For Slurm FavorSmall all the jobs have a wait time of 0 and all the jobs of the experiment are launched. The average utilization is the same as OAR, 25%.

OAR experiment without the setting of the context shows a higher wait time than Slurm, this confirms the fact that Slurm is still faster than OAR to process the jobs.

It is particularly remarkable that the reconstruction of the context has a very strong impact on the replay results. Without setting up the context, the platform has a low utilization and thus the wait times and slowdown are falsely decreased. The problem of replaying a trace out of its context is that we analyze the warm-up period instead of the steady state. This means that we analyze the capability of the platform to load and not the capability to treat an on-line workload.

This shows that when using a workload model to evaluate a RJMS system, either the workload has to be more loaded, or the experiment has to be longer in order to pass from the warm-up period to the steady state during the replay itself. The difficulty here is to define what should be "more loaded" and "longer". This is dependent of the workload from which the model was derived and thus this information is not easily available for the end-user experimenter.

An alternative of using the context reconstruction would be to replay a longer trace and take into consideration only the end of the replay. However this is less efficient and is actually not exactly valid. First, regarding the time of the experiment, the context reconstruction takes only few minutes, according to the RJMS job submission processing time. A full run of the jobs in the system before the extract would inevitably be longer than that, and the submission processing time would be still present here. Then, if we do that, we do not replay a workload on a "system snapshot". All the jobs replayed in the experiment's system are scheduled according to its policies, this means that if we replay a longer workload, the schedule would be different and the system state at the start of the trace extract replay would not be the same. For these two reasons we insist on the use of the context as we describe it when doing workload trace replay.

4.2 On the Interest of Replaying a Real Trace

The ESP benchmark [118] presented in Section 1.1 is a RJMS benchmarking tool enabling to evaluate such a system by giving a global performance metric. This metric is the workload makespan ratio over the workload length; i.e. the time necessary to the system to process the full workload, from the first job submission to the last job ending, over the total area of the workload. The closer to 1 is this ratio, the best is the RJMS performance.

In its first version the ESP benchmark is composed of a set of 82 jobs. 80 of these arrive in three batches separated by 10 minutes. The jobs attribution to a given batch is random. The other two jobs arrive after these three batches and request for the full system. This benchmark is the only benchmark available that is dedicated to RJMS evaluation. It is used in several works as [117] and discussed in [77, 39].

The ESP benchmark was later rebuilt and improved in its second version (v2) that corrected several of the first version weak points, as the problem of the booting period that was unrealistic, or the division of jobs in only three batches. The second version was used in [14] and [54] and is more realistic. ESP v2 uses a two-stage workload submission. The first stage is the throughput mode where the benchmark fills the cluster with jobs that arrive sequentially at short regular interval. By default 50 jobs are submitted during this phase. Then after a pause, the rest of the jobs are submitted regarding a Gaussian arrival process with a mean of 6 and standard deviation of 1. This means that jobs inter-arrivals are very close. The total number of jobs submitted is 230 and a full ESP v2 runs in about three hours. To determine jobs characteristics, a given job is attributed to one of the 14 job classes, each corresponding to a proportion of requested resources and a runtime. Each job class has a fixed number of jobs. The jobs of the "Z" class request the whole cluster resources and have a higher priority than the other classes. Once a job has been attributed to a class it is not deterministic to know if it will take part of the first submission stage (throughput arrival mode) or the second one (random arrival mode) as the jobs submission sequence is randomized. The advantage of this approach is that the second phase of the workload submission does not start on an empty system. The drawback is that it generates in its first stage a non realistic context of jobs submitted in throughput mode to fill the cluster, then a second stage where the jobs arrive regarding a Gaussian arrival process. These two stages are totally different in terms of impact on the RJMS and should not be considered at the same level for the RJMS evaluation.

An other important drawback is the fact that the jobs model in ESP does not consider the user request of time (walltime). As we have seen in our experiments this impacts a lot the ability for a given configuration to use backfilling techniques depending on the weight put on this characteristic. The ability to backfill small jobs has a strong impact on the system utilization and thus on the makespan ratio used as a performance metric in ESP v2.

However, despite these drawbacks, the ESP v2 benchmark is a very convenient and easy to set up tool that enables to rapidly give a performance score to a given configuration. A good tuning of the jobs repartition among the classes based on

a sophisticated workload model (as the Feitelson98, Lublin99 and Cirne01 models, available in the PWA[1], see Part I Section 1.2) might correct the lack of realism of the underlying model and with a realistic arrival process during the second submission stage the workload model would be stronger. An important improvement though, would be to not account for the jobs submitted in the throughput mode in the final evaluation score. The consideration of these jobs in the score creates a bias where the warm-up period is taken into account in the evaluation. As we have seen, this has a strong impact on the performance results to start the replay on an empty system.

With the use of a real trace extract replaced in its context, the problematic is slightly different. The realism of the experiment depends on the trace extract that is chosen for the experiment. The use of real trace suffers from the same problem as the use of workload models: the choice the experimenter has to face. In models the experimenter has to choose the model itself, and select the parameters to apply to the model. This choice is not easy and will determine the experiment value. A good model with bad input parameters can lead to a workload unadapted to the targeted platform. With the use of a real workload trace within its original context, we replay a given scenario. As the trace is taken from a real case that occurred on a production platform, we know that we are replaying something that makes sense. At least it made sense in the original platform from where and when the trace was extracted. Now, it belongs to the experimenter to take the trace replay for what it is: the response of a given system to a given stress. The stress will impact differently platforms of different sizes and configurations. A workload extract, replayed in a platform comparable to the one from where it was extracted, is a powerful tool to test the system response on the given configuration. But the scope of the replay is not only limited to this. A replay can give insights on the system response to a large failure with the replay on a sub-part of the platform. A trace can also be scaled to fit a different platform configuration, as proposed in [37]. And a trace with a particular pattern may be used to test the best configuration of the system to a given scenario as used in Section 3.3.

However, the replay of a trace as we propose it, has several drawback. First the replay disrupts the *user feedback effect*, largely discussed in [41]. This effect comes from the observation of the system state by the users and their job submission according to this state. This has led to the idea of "Think-Time" [16] described in D. Feitelson's works. Then, our approach currently does not support failures and downtimes in the replay. We are currently working on these aspects.

When we replay a trace, we replay a short trace extract, on the order of the hour or few hours. This limits a lot the failure effect as, the shorter the trace is, the lower is the probability to have a failure. For a one hour trace this probability is almost negligible. However as a starting point, we plan to use failure models [96, 74] to couple to the experimental replay in order to take failures into account. Then, when workload logs with the information on failures will become available it will be

[1]PWA: http://www.cs.huji.ac.il/labs/parallel/workload

possible to go deeper in the reconstruction of the trace context, where it will not be only determined by jobs, but also on machine states.

Concerning the user feedback effect the first problem we encounter is the lack of information on Think-Times in many of the PWA logs. If the RJMS from where the log is extracted does not provide the information on jobs dependencies, this information have to be calculated. Calculating users Think-Times is a difficult problem that can be summarized by the session and batch detection problem in workload logs [5, 4]. A *batch*, as defined in [5], is composed of a set of independent jobs submitted by a user over a short period of time. This is also sometimes referred as campaigns [89, 33, 69]. Still in [5], the authors propose several methods to detect batches within a user session and evaluate them. However the users session/batch detection techniques described in these works cannot lead to globally deterministic results as they rely on a calculated threshold that is dependent on the users behavior on the platform. This is probably because in many cases the job dependencies are not known directly by the RJMS, and because of the difficulty to calculate this information post-mortem, that a lot of parallel logs from PWA miss it.

Moreover, our current version of *riplay* does not take into account user Think-Times and jobs dependencies. The submission of the jobs follows strictly the original trace. However we are currently working on a newer version that could take account of this information to add a synthetic user feedback effect into the replayed trace.

Discussion on the *Representativeness* of a Trace Extract

With the workload replay technique we tend to want to replay a *representative* trace that has characteristics of the average behavior of the jobs in the system. However it might also be interesting to use workload traces with very particular and well identified patterns, such as the workload "flurries" [51, 45]: *rare surges of activity with a repetitive nature, caused by a single user, that dominate the workload for a relatively short period* ([113]). In [50] the authors propose to treat these flurries separately from the rest of the workload and to not consider them in the derived model and in the SWF trace distributed in the PWA. However they also propose the uncleaned trace as the study of flurries in workload is an interesting aspect of the work [113]. Flurries may strongly impact locally the RJMS as they generate a high load on the system. The experimental study of the system's response to flurries is primordial to better understand the performance bottlenecks. In [109] it was also discovered that although the flurries have a big impact on the workload, they are not the only cause of disturbance. This opens the door to evaluating other kinds of patterns to better understand their impact on the system.

Chapter 5

Conclusions and Future Research Directions

5.1 Conclusions

High Performance Computing platforms have rapidly evolved in both terms of size and performance. From the hardware constantly more efficient and the applications always demanding for more computing power, to the RJMS that juggles to manage their orchestration. We are now preparing the era of Exascale. With such a progress in terms of computing capacity, we are facing new scalability problems and may have to re-think our approach, such as the recent questions on how to manage Big Data [111]. The observation and analysis of the users workload has been a tool for performance analysis for many years. Now we have to go deeper in the analysis and consider the system in its whole. It is through a deep study of the workload that we will better understand nowadays platform uses, and it is from this study that we will understand our new future challenges.

The research work presented in the first part of this thesis aims to initiate this movement by the proposal of an extended workload trace format, based on the study of existing formats and our experience in analysis of several workloads. The format proposed comes on top of the existing reference format, SWF to describe more workload features. The goal of this format proposal is the opening to evolution and customization. This was one of the main criteria for the design of the proposal. We analyzed several production cluster workload traces and compare them regarding commonly used performance metrics. Through this analysis we were able to characterize the global behaviors of the workloads from the different platforms We also analyzed in depth resource consumptions by the jobs on two of these clusters. This enabled to detect several abnormal behaviors and to propose scheduling enhancements to respond to these problems.

In the second part, we have presented a methodology accompanied with a set of tools to experiment the replay of a selected trace extract, replaced in its original context. This study enabled to evaluate the scalability of a new scheduler for OAR

RJMS and to compare two scheduling policies for an other RJMS, Slurm. Based on this study, we were able to improve OAR scheduler. It is now able to manage a load from a platform of a size and workload density such as Curie. For Slurm, we discovered that the default *multifactor* scheduling policy put too much priority weight on large jobs. This results in a low system utilization and few jobs being processed when the user run times estimates are too big. We also observed that the option proposed to counter this phenomenon and that favors small jobs, actually solves this problem but leads to such an unfairness that large jobs slowdowns were too big. This shows that there is still work to do in the fine tuning of jobs weights in Slurm's *multifactor* policy and that the option to favor small jobs should be configurable, as it currently puts too much weight on small jobs.

The coupling between workload analysis of the Curie platform and experimental replay in a reproducible reconstruction of this platform on top of Grid'5000, enabled us to test configuration changes and observe system response to this change. This would not have been possible in the production platform as it would have forced us to stop production and thus to disturb the users.

This approach will enable us later to pursue experimentally the search for the harmony of system configuration to users workload through a thorough selection of workload patterns, identified as problematic for the system. This study will be useful to other researches, such as [25] that propose to dynamically change the type of scheduling regarding the input workload type.

The research work presented in this dissertation also resulted in the following publications [34, 36, 35, 33] and to the *evalys-tools* repository that aims at collecting and sharing knowledge and tools on workload traces analysis and experimentation.

With the in depth study and understanding of the workload characteristics, we will be able to better understand future needs. This will help us to better build the road to Exascale.

5.2 Future Research Directions

Concerning future directions, the first step will be to start populating the *evalys-tools* repository with collected data from workload, job consumptions and node failures, in the workload format proposed. This will enable to continue the process of workload analysis, with a multiple and richer information. In a second time, this will allow us to have a feedback on the format itself and thus to improve it. It will be through the regular usage that we will see the future directions of the format.

The second step will be the characterization of jobs resource consumptions. We plan to use a time series approach to detect patterns in the resource usages and try to classify the jobs regarding their type of consumption. The temporal characterization of cpu, memory, I/O and network usage will also be the starting point of a classification <*user, code*> to enable a finer workload management based on the knowledge of the behavior of the users and their respective types of job.

Concerning the experimental approach, we will improve the *riplay* tool to support the test of fairshare scheduling techniques, and improve the trace replay experiment methodology, to take into account failures from the logs if available, or from model otherwise. This will allow us to recreate more finely a real production platform environment for the replay.

Future research directions also include the extension of the replay methodology with synthetic jobs created from the analysis and classification of jobs consumptions. By this mean we want the replay to test other scheduling features in the platform, such as the declaration by the user to the RJMS, of the signature of the job. At first, this signature can be simply the type of job to run: cpu, memory or I/O bound for example. This will allow to study the impact of postponing, at scheduling level, some of the jobs that do parallel I/O on the file system.

Finally, by the mean of emulation technique along with the trace replay, as used in our works upon Grid'5000, it would be possible to try Exascale scalability benchmarking of the RJMS. To do this, we will need an Exascale machine trace and thus, need the help of workload models to imagine such a trace.

Appendix

Acronyms

BoT Bag of Tasks 22

BLAS Basic Linear Algebra Subprograms 3

CDF Cumulated Distribution Function 30

DAG Direct Acyclic Graph 22

DCG Direct Cyclic Graph 22

DFS Distributed File System 1

FLOPS FLoating point Operations Per Second 2

FTA Failure Trace Archive 24

GCT Google Cluster-usage Traces 18

GPU Graphics Processing Unit 2

GPGPU General-Purpose Graphics Processing Unit 2

GWA Grid Workload Archive 22

GWF Grid Workload Format 22

HPC High Performance Computing 1

MPI Message Passing Interface 3

PDF Probability Density Function 33

PFLOPS PetaFLOPS 35

PWA Parallel Workload Archive 6

QOS Quality Of Service 2

RJMS Resource and Job Management System 1

ROI Return on Investment 4

SWF Standard Workload Format 16

VSIPL Vector Signal Image Processing Library 3

Bibliography

[1] http://www.cs.huji.ac.il/labs/parallel/workload/logs.html.

[2] *Final Report of the NSF Workshop on Challenges of Scientific Workflows*, National Science Foundation, Arlington, VA, December, 2006.

[3] *CDE: Run Any Linux Application On-Demand Without Installation*. Proceedings of the 2011 USENIX Large Installation System Administration Conference (LISA), 2011.

[4] *On Extracting Session Data from Activity Logs*, 5th Ann. Intl. Conf. Systems & Storage, June 2012.

[5] *On identifying user session boundaries in parallel workload logs*, Job Scheduling Strategies for Parallel Processing, May 2012.

[6] C. Bailey Lee, Y. Schwartzman, J. Hardy, and A. Snavely. Are user runtime estimates inherently inaccurate? In D. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 3277 of *Lecture Notes in Computer Science*, pages 253–263. Springer Berlin Heidelberg, 2005.

[7] M. A. Baker, G. C. Fox, and H. W. Yau. Cluster computing review, 1995.

[8] G. Berry. Preemption in concurrent systems. In *Proceedings of FSTTCS*, 1993.

[9] F. Z. Boito, R. V. Kassick, and P. O. Navaux. The impact of applications' i/o strategies on the performance of the lustre parallel file system. *International Journal of High Performance Systems Architecture*, 3(2):122–136, 2011.

[10] P. Buneman, S. Khanna, and W.-C. Tan. Data provenance: Some basic issues. In S. Kapoor and S. Prasad, editors, *FST TCS 2000: Foundations of Software Technology and Theoretical Computer Science*, volume 1974 of *Lecture Notes in Computer Science*, pages 87–93. Springer Berlin / Heidelberg, 2000.

[11] P. Buneman, S. Khanna, and T. Wang-Chiew. Why and where: A characterization of data provenance. In J. Van den Bussche and V. Vianu, editors, *Database Theory - ICDT 2001*, volume 1973 of *Lecture Notes in Computer Science*, pages 316–330. Springer Berlin / Heidelberg, 2001.

[12] M. Calzarossa and G. Serazzi. A characterization of the variation in time of workload arrival patterns. *Computers, IEEE Transactions on*, C-34(2):156–162, Feb.

[13] J. Cao and F. Zimmermann. Queue scheduling and advance reservations with cosy. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pages 63–, April.

[14] N. Capit, G. D. Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounie, P. Neyron, and O. Richard. A batch scheduler with high level components. *Cluster Computing and the Grid*, pages 776–783, 2005.

[15] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quetier, and O. Richard. Grid'5000: A large scale and highly reconfigurable grid experimental testbed. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, GRID '05, pages 99–106, Washington, DC, USA, 2005. IEEE Computer Society.

[16] S. J. Chapin, W. Cirne, D. G. Feitelson, J. P. Jones, S. T. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby. Benchmarks and standards for the evaluation of parallel job schedulers. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 67–90. Springer-Verlag, 1999. Lect. Notes Comput. Sci. vol. 1659.

[17] F. Chen and S. Majumdar. Performance of parallel i/o scheduling strategies on a network of workstations. In *Parallel and Distributed Systems, 2001. ICPADS 2001. Proceedings. Eighth International Conference on*, pages 157–164, 2001.

[18] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33:3–12, July 2003.

[19] W. Cirne and F. Berman. Adaptive selection of partition size for supercomputer requests. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1911 of *Lecture Notes in Computer Science*, pages 187–207. Springer Berlin Heidelberg, 2000.

[20] W. Cirne and F. Berman. A comprehensive model of the supercomputer workload. In *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pages 140–148, Dec.

[21] E. F. Codd. Further normalization of the data base relational model. *Data base systems*, pages 33–64, 1972.

[22] R. Curry and R. Simmonds. Job centric cluster monitoring. In *Parallel and Distributed Systems, 2006. ICPADS 2006. 12th International Conference on*, volume 1, page 8 pp., 0-0 2006.

[23] C. J. Date. *An introduction to database systems*. Addison Wesley Publishing Company, 1986.

[24] T. Delaet, W. Joosen, and B. Vanbrabant. A survey of system configuration tools. In *Proceedings of the 24th international conference on Large installation system administration*, LISA'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

[25] K. Deng, R. Verboon, K. Ren, and A. Iosup. A periodic portfolio scheduler for scientific computing in the data center. In E. F. W. Cirne, N. Desai and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, Lecture Notes in Computer Science. Springer, 2013.

[26] J. J. Dongarra, P. Luszczek, and A. Petitet. The linpack benchmark: past, present and future. *Concurrency and Computation: Practice and Experience*, 15(9):803–820, 2003.

[27] A. Downey. Using queue time predictions for processor allocation. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science*, pages 35–57. Springer Berlin Heidelberg, 1997.

[28] A. Downey. Predicting queue times on space-sharing parallel computers. In *Parallel Processing Symposium, 1997. Proceedings., 11th International*, pages 209–218, Apr.

[29] A. Downey. A parallel workload model and its implications for processor allocation. In *High Performance Distributed Computing, 1997. Proceedings. The Sixth IEEE International Symposium on*, pages 112–123, Aug.

[30] A. B. Downey and D. G. Feitelson. The elusive goal of workload characterization. *SIGMETRICS Perform. Eval. Rev.*, 1999.

[31] T. A. El-Ghazawi, K. Gaj, N. A. Alexandridis, F. Vroman, N. Nguyen, J. R. Radzikowski, P. Samipagdi, and S. A. Suboh. A performance study of job management systems. *Concurrency - Practice and Experience*, 2004.

[32] R. Elmasri and S. Navathe. *Fundamentals of database systems*. 2009.

[33] J. Emeras, V. Pinheiro, K. Rzadca, and D. Trystram. Ostrich: Fair scheduling for multiple submissions. In *PPAM'2013*, 2013.

[34] J. Emeras, O. Richard, and B. Bzeznik. Reconstructing the Software Environment of an Experiment with Kameleon. In *ACM Compute*. ACM India, jan 2012.

[35] J. Emeras, C. Ruiz, J.-M. Vincent, and O. Richard. Analysis of the jobs resource utilization on a production system. In E. F. W. Cirne, N. Desai and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, Lecture Notes in Computer Science. Springer, 2013.

[36] J. Emeras, C. Ruiz, J.-M. Vincent, and O. Richard. Jobs resource utilization as a metric for clusters comparison and optimization. In *ComPAS'2013*, 2013.

[37] C. Ernemann, B. Song, and R. Yahyapour. Scaling of workload traces. In D. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 2862 of *Lecture Notes in Computer Science*, pages 166–182. Springer Berlin / Heidelberg, 2003.

[38] D. Feitelson. Packing schemes for gang scheduling. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1162 of *Lecture Notes in Computer Science*, pages 89–110. Springer Berlin Heidelberg, 1996.

[39] D. Feitelson. A critique of esp. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1911 of *Lecture Notes in Computer Science*, pages 68–73. Springer Berlin Heidelberg, 2000.

[40] D. Feitelson. Metrics for parallel job scheduling and their convergence. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 2221 of *Lecture Notes in Computer Science*, pages 188–205. Springer Berlin Heidelberg, 2001.

[41] D. Feitelson. Workload modeling for performance evaluation. In M. Calzarossa and S. Tucci, editors, *Performance Evaluation of Complex Systems: Techniques and Tools*, volume 2459 of *Lecture Notes in Computer Science*, pages 114–141. Springer Berlin / Heidelberg, 2002.

[42] D. Feitelson. Experimental analysis of the root causes of performance evaluation results: a backfilling case study. *Parallel and Distributed Systems, IEEE Transactions on*, 16(2):175–182, Feb.

[43] D. Feitelson and M. Jettee. Improved utilization and responsiveness with gang scheduling. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science*, pages 238–261. Springer Berlin Heidelberg, 1997.

[44] D. Feitelson and L. Rudolph. Metrics and benchmarking for parallel job scheduling. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1459 of *Lecture Notes in Computer Science*, pages 1–24. Springer Berlin Heidelberg, 1998.

[45] D. Feitelson and D. Tsafrir. Workload sanitation for performance evaluation. In *Performance Analysis of Systems and Software, 2006 IEEE International Symposium on*, pages 221–230, 2006.

[46] D. Feitelson and A. Weil. Utilization and predictability in scheduling the ibm sp2 with backfilling. In *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International ... and Symposium on Parallel and Distributed Processing 1998*, pages 542–546, Mar-3 Apr.

[47] D. G. Feitelson. A critique of ESP. In D. G. Feitelson and L. Rudolph, editors, *JSSPP*, volume 1911 of *Lecture Notes in Computer Science*, pages 68–73. Springer, 2000.

[48] D. G. Feitelson. Metric and workload effects on computer systems evaluation. *Computer*, 36(9):18–25, Sept. 2003.

[49] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong. Theory and practice in parallel job scheduling. In *JSSPP*, 1997.

[50] D. G. Feitelson, D. Tsafrir, and D. Krakov. Experience with the parallel workloads archive. 2012.

[51] E. Frachtenberg and D. G. Feitelson. Pitfalls in parallel job scheduling evaluation. In *JJSSPP*, 2005.

[52] K. Fuerlinger, N. J. Wright, and D. Skinner. Effective performance measurement at petascale using ipm. In *Proceedings of The Sixteenth IEEE International Conference on Parallel and Distributed Systems (ICPADS 2010)*, Shanghai, China, Dec. 2010.

[53] J. Gehring and T. Preiss. Scheduling a metacomputer with uncooperative sub-schedulers. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1659 of *Lecture Notes in Computer Science*, pages 179–201. Springer Berlin Heidelberg, 1999.

[54] Y. Georgiou. *Contributions for Resource and Job Management in High Performance Computing*. PhD thesis, LIG, Grenoble - France, Sep 2010.

[55] Y. Georgiou and M. Hautreux. Evaluating scalability and efficiency of the resource and job management system on large hpc clusters. In *JSSPP*. 2012.

[56] Y. Georgiou, J. Leduc, B. Videau, J. Peyrard, and O. Richard. A tool for environment deployment in clusters and light grids. In *Second Workshop on System Management Tools for Large-Scale Parallel Systems (SMTPS'06)*, Rhodes Island, Greece, April 2006.

[57] Y. Georgiou, J. Leduc, B. Videau, J. Peyrard, and O. Richard. A tool for environment deployment in clusters and light grids. In *IPDPS*. IEEE, 2006.

[58] Y. Georgiou, O. Richard, and N. Capit. Evaluations of the lightweight grid cigri upon the grid5000 platform. In *E-SCIENCE '07: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*. IEEE Computer Society, 2007.

[59] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers. Examining the challenges of scientific workflows. *Computer*, 40(12):24 –32, 2007.

[60] S. Gunther, M. Haupt, and M. Splieth. Utilizing internal domain-specific languages for deployment and maintenance of it infrastructures, fin-004-2010. Technical report, University of Magdeburg, School of Computer Science.

[61] J. GUSTEDT, E. JEANNOT, and M. QUINSON. Experimental methodologies for large-scale systems: A survey. *Parallel Processing Letters*, 19(03):399–418, 2009.

[62] P. Hudak. Modular domain specific languages and tools. In *Proceedings of the 5th International Conference on Software Reuse*, ICSR '98, pages 134–, Washington, DC, USA, 1998. IEEE Computer Society.

[63] E. Imamagic and D. Dobrenic. Grid infrastructure monitoring system based on nagios. In *Proceedings of the 2007 workshop on Grid monitoring*, GMW '07, pages 23–28, New York, NY, USA, 2007. ACM.

[64] A. Iosup. *A framework for the study of grid inter-operation mechanisms*. PhD thesis, Delft University of Technology, 2009.

[65] A. Iosup, M. Jan, O. Sonmez, and D. Epema. On the dynamic resource availability in grids. In *Grid Computing, 2007 8th IEEE/ACM International Conference on*, pages 26 –33. sept. 2007.

[66] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. H. Epema. The grid workloads archive. *Future Generation Computer Systems*, 24(7):672 – 686, 2008.

[67] D. Jackson, Q. Snell, and M. Clement. Core algorithms of the maui scheduler. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 2221 of *Lecture Notes in Computer Science*, pages 87–102. Springer Berlin Heidelberg, 2001.

[68] R. Jain. *The Art of Computer Systems Performance Analysis: techniques for experimental design, measurement, simulation, and modeling*. Wiley, 1991.

[69] R. Jamet and G. Huard. Task aggregation in cigri, a grid management middleware.

[70] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riordan. Modeling of workload in mpps. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science*, pages 95–116. Springer Berlin Heidelberg, 1997.

[71] J. A. Kaplan and M. L. Nelson. A comparison of queueing, cluster and distributed computing systems. Technical report, NASA Langley Research Center, 1994.

[72] C. Klein and C. Perez. An rms for non-predictably evolving applications. In *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, pages 326–334, 2011.

[73] C. Klein and C. Pérez. Towards scheduling evolving applications. In M. Alexander, P. D'Ambra, A. Belloum, G. Bosilca, M. Cannataro, M. Danelutto, B. Martino, M. Gerndt, E. Jeannot, R. Namyst, J. Roman, S. Scott, J. Traff, G. Vallée, and J. Weidendorfer, editors, *Euro-Par 2011: Parallel Processing Workshops*, volume 7155 of *Lecture Notes in Computer Science*, pages 117–127. Springer Berlin Heidelberg, 2012.

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

[75] A. Kopytov. Sysbench: a system performance benchmark, 2004.

[76] D. Krakov and D. G. Feitelson. High-resolution analysis of parallel job workloads. In E. F. W. Cirne, N. Desai and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 7698 of *Lecture Notes in Computer Science*, pages 178–195. Springer-Verlag, 2012.

[77] W. T. Kramer. *PERCU: A Holistic Method for Evaluating High Performance Computing Systems*. PhD thesis, EECS Department, University of California, Berkeley, Nov 2008.

[78] B. Lawson and E. Smirni. Multiple-queue backfilling scheduling with priorities and reservations for parallel systems. In D. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 2537 of *Lecture Notes in Computer Science*, pages 72–87. Springer Berlin Heidelberg, 2002.

[79] B. Lawson, E. Smirni, and D. Puiu. Self-adapting backfilling scheduling for parallel systems. In *Parallel Processing, 2002. Proceedings. International Conference on*, pages 583–592, 2002.

[80] C. B. Lee and A. Snavely. On the user–scheduler dialogue: Studies of user-provided runtime estimates and utility functions. *International Journal of High Performance Computing Applications*, 2006.

[81] W. Leland and T. J. Ott. Load-balancing heuristics and process behavior. *SIGMETRICS Perform. Eval. Rev.*, 14(1):54–69, May 1986.

[82] U. Lublin and D. G. Feitelson. The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing*, 63:2003, 2001.

[83] M. Margo, K. Yoshimoto, P. Kovatch, and P. Andrews. Impact of reservations on production job scheduling. In E. Frachtenberg and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 4942 of *Lecture Notes in Computer Science*, pages 116–131. Springer Berlin Heidelberg, 2008.

[84] M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: Design, implementation and experience, 2004.

[85] A. Mu'alem and D. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. *Parallel and Distributed Systems, IEEE Transactions on*, 12(6):529–543, 2001.

[86] J. S. Mustafa Uysal, Anurag Acharya. Maryland applications for measurement and benchmariking of i/o on parallel computers, May 1997. http://www.cs.umd.edu/projects/hpsl/mambo/.

[87] A. Nataraj, M. Sottile, A. Morris, A. Malony, and S. Shende. Tauoversupermon: Low-overhead online parallel performance monitoring, 2007.

[88] J. K. Ousterhout. Scheduling techniques for concurrent systems. In *ICDCS*, 1982.

[89] V. Pinheiro, K. Rzadca, and D. Trystram. Campaign scheduling. In *IEEE International Conference on High Performance Computing (HiPC), Proceedings*, 2012.

[90] C. Reiss, J. Wilkes, and J. L. Hellerstein. Google cluster-usage traces: format + schema. Technical report, Google Inc., Mountain View, CA, USA, Nov. 2011. Revised 2012.03.20. Posted at URL `http://code.google.com/p/googleclusterdata/wiki/TraceVersion2`.

[91] C. Reiss, J. Wilkes, and J. L. Hellerstein. Obfuscatory obscanturism: making workload traces of commercially-sensitive systems safe to release. In *3rd International Workshop on Cloud Management (CLOUDMAN'12)*, pages 1279–1286. IEEE, Apr. 2012.

[92] E. Rosti, G. Serazzi, E. Smirni, and M. S. Squillante. The impact of i/o on program behavior and parallel scheduling. In *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, SIGMETRICS '98/PERFORMANCE '98, pages 56–65, New York, NY, USA, 1998. ACM.

[93] L. Rudolph and P. Smith. Valuation of ultra-scale computing systems. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1911 of *Lecture Notes in Computer Science*, pages 39–55. Springer Berlin / Heidelberg, 2000. 10.1007/3-540-39997-6_3.

[94] L. Rudolph and P. Smith. Valuation of ultra-scale computing systems. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1911 of *Lecture Notes in Computer Science*, pages 39–55. Springer Berlin / Heidelberg, 2000.

[95] B. Schroeder and G. A. Gibson. A large scale study of failures in high-performance-computing systems. In *International Symposium on Dependable Systems and Networks (DSN)*, June 2006.

[96] B. Schroeder and G. A. Gibson. A large-scale study of failures in high-performance computing systems. *Dependable Systems and Networks, International Conference on*, 0:249–258, 2006.

[97] H. Shan, K. Antypas, and J. Shalf. Characterizing and predicting the i/o performance of hpc applications using a parameterized synthetic benchmark. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 42:1–42:12, Piscataway, NJ, USA, 2008. IEEE Press.

[98] H. Shan and J. Shalf. Using ior to analyze the i/o performance for hpc platforms. *Cray User Group Conference, Seattle, WA, May7-10, 2007*, 2007.

[99] S. Sharma, P. G. Bridges, and A. B. Maccabe. A framework for analyzing linux system overheads on hpc applications. In *Proceedings of the 2005 Los Alamos Computer Science Institute Symposium*, October 2005.

[100] S. S. Shende and A. D. Malony. The tau parallel performance system. *The International Journal of High Performance Computing Applications*, 20:287–331, 2006.

[101] Y. L. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Rec.*, 34:31–36, September 2005.

[102] J. Skovira, W. Chan, H. Zhou, and D. A. Lifka. The easy - loadleveler api project. In *JSSPP*, 1996.

[103] W. Smith, I. Foster, and V. Taylor. Scheduling with advanced reservations. In *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*, pages 127–132, 2000.

[104] Q. Snell, M. Clement, D. Jackson, and C. Gregory. The performance impact of advance reservation meta-scheduling. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1911 of *Lecture Notes in Computer Science*, pages 137–153. Springer Berlin Heidelberg, 2000.

[105] B. Song, C. Ernemann, and R. Yahyapour. Parallel computer workload modeling with markov chains. In D. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2005.

[106] M. J. Sottile and R. G. Minnich. Supermon: A high-speed cluster monitoring system. In *Proceedings of the IEEE International Conference on Cluster Computing*, CLUSTER '02, Washington, DC, USA, 2002. IEEE Computer Society.

[107] A. Streit. A self-tuning job scheduler family with dynamic policy switching. In *JSSPP*, 2002.

[108] M. Szomszor and L. Moreau. Recording and reasoning over data provenance in web and grid services. In *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, volume 2888 of *Lecture Notes in Computer Science*, pages 603–620. Springer Berlin / Heidelberg, 2003.

[109] D. Talby and D. Feitelson. Improving and stabilizing parallel computer performance using adaptive backfilling. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 84a–84a, 2005.

[110] D. Talby and D. G. Feitelson. Supporting priorities and improving utilization of the IBM SP scheduler using slack-based backfilling. In *IPPS/SPDP*, 1999.

[111] O. Trelles, P. Prins, M. Snir, and R. C. Jansen. Big data, but are we ready? *Nature reviews Genetics*, 12(3):224–224, 2011.

[112] D. Tsafrir, Y. Etsion, and D. Feitelson. Modeling user runtime estimates. In D. Feitelson, E. Frachtenberg, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 3834 of *Lecture Notes in Computer Science*, pages 1–35. Springer Berlin Heidelberg, 2005.

[113] D. Tsafrir and D. Feitelson. Instability in parallel job scheduling simulation: the role of workload flurries. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 10 pp.–, 2006.

[114] G. von Laszewski, G. Fox, F. Wang, A. Younge, A. Kulshrestha, G. Pike, W. Smith, J. Vockler, R. Figueiredo, J. Fortes, and K. Keahey. Design of the futuregrid experiment management framework. In *Gateway Computing Environments Workshop (GCE), 2010*, pages 1–10, 2010.

[115] A. M. Weil and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Trans. Parallel Distrib. Syst*, 2001.

[116] J. Weinberg and A. Snavely. User-guided symbiotic space-sharing of real workloads. In *Proceedings of the 20th annual international conference on Supercomputing*, ICS '06, pages 345–352, New York, NY, USA, 2006. ACM.

[117] A. Wong, L. Oliker, W. Kramer, T. Kaltz, and D. Bailey. System utilization benchmark on the cray t3e and ibm sp. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1911 of *Lecture Notes in Computer Science*, pages 56–67. Springer Berlin Heidelberg, 2000.

[118] A. Wong, L. Oliker, W. T. C. Kramer, T. Kaltz, and D. Bailey. Esp: A system utilization benchmark. In *Supercomputing, ACM/IEEE 2000 Conference*, 2000.

[119] A. Yoo, M. Jette, and M. Grondona. Slurm: Simple linux utility for resource management. In D. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 2862 of *Lecture Notes in Computer Science*, pages 44–60. Springer Berlin / Heidelberg, 2003.

[120] N. Zakay and D. G. Feitelson. Workload resampling for performance evaluation of parallel job schedulers. In *4th Intl. Conf. Performance Engineering*. 2013.

[121] Y. Zhang, A. Sivasubramaniam, J. Moreira, and H. Franke. Impact of workload and system parameters on next generation cluster scheduling mechanisms. volume 12, pages 967–985, 2001.