Université de Grenoble

Thèse

Pour obtenir le grade de

Docteur de l'Université de Grenoble
Spécialité : Informatique

Arrêté ministériel : 7 août 2006

Présentée et soutenue publiquement par

Yiannis Georgiou

le 5 novembre 2010

Contributions à la Gestion de Ressources et de Tâches pour le Calcul de Haute Performance

---

Thèse dirigée par Olivier Richard et codirigée par Jean-Francois Méhaut

Jury

Frank	Cappello	Directeur de Recherche à INRIA, France	Rapporteur
William	Kramer	Directeur de Recherche à NCSA, USA	Rapporteur
Morris	Jette	Informaticien au LLNL, USA	Examinateur
Pascale	Rosse-Laurent	Architecte Informatique à BULL, France	Examinatrice
Daniel	Hagimont	Professeur à INPT/ENSEEIHT, France	Président
Olivier	Richard	Maître de Conférence à l'UJF	Directeur
Jean-Francois	Méhaut	Professeur à l'UJF, France	Co-Directeur

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

CONTRIBUTIONS FOR

RESOURCE AND JOB MANAGEMENT IN HIGH PERFORMANCE COMPUTING

CONTRIBUTIONS FOR
RESOURCE AND JOB MANAGEMENT IN HIGH PERFORMANCE COMPUTING

A Dissertation
submitted to the department of
Computer Science
and the committee on graduate studies of
Joseph Fourier University
in partial fulfillment of the requirements
for the degree of
Doctor Of Philosophy

Yiannis Georgiou

September 2010

Abstract

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 HPC since it has a strategic place in the whole software stack because it stands between the above two layers. However the latest evolutions in hardware and applications layers have provided new levels of complexities to this middleware. Issues like scalability, management of topological constraints, energy efficiency and fault tolerance have to be particularly considered, among others, in order to provide a better system exploitation from both the system and user point of view.

This dissertation provides a state of the art upon the fundamental concepts and research issues of Resources and Jobs Management Systems. It provides a multi-level comparison (concepts, functionalities, performance) of some of the most wideley used Resource and Jobs Management Systems in High Performance Computing.

An important metric to evaluate the work of a RJMS on a platform is the observed system utilization. However studies and logs of production platforms show that HPC systems in general suffer of significant unutilization rates. Our study deals with these clusters' unutilization periods by proposing methods to aggregate otherwise unutilized resources for the benefit of the system or the application. More particularlly this thesis explores RJMS level mechanisms: 1) for increasing the jobs valuable computation rates in the high volatile environments of a lighweight grid context, 2) for improving system utilization with malleability techniqes and 3) providing energy efficient system management through the exploitation of idle computing machines.

The experimentation and evaluation in this type of contexts provide important complexities due to the inter-dependency of multiple parameters that have to be taken into control. In our study we have developped a methodology based upon real-scale controlled experimentation with submission of synthetic or real workload traces.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

High performance computing is characterized by the growing evolution of computing architectures, the rapid proliferation of computing resources, and the increasing complexity of problems, users wish to solve. The latest evolution on multiprocessor technologies presented the dawn of the multicore period. This new period of computing, enforced scientists with a huge computing power, but also a new level of complexity. Within the last 11 years, we observe a performance increase from Gigaflops (Intel ASCI Red in 1997) up to the Petaflops (IBM Roadrunner in 2008) by a factor of 1000 [3]. In the same time a great deal of parallel compute-intensive applications have been developed to carry out important scientific research. The applications may span all different areas of science like astrophysics, biology, medicine, climate modeling, weather prediction up to crash simulations, image rendering and film processing. The large diversity of computing platforms: clusters, multiclusters, grids, lightweight grids and lately clouds, provide the means for all kind of scientific applications to perform computations.

In order to efficiently deliver computing power to multiple users at the same time, specialised softwares are responsible for the matching of user jobs processing needs with the available resources. These software solutions, referred as Resource and Job Management Systems (RJMS), provide functions for building, submitting, processing and monitoring jobs quickly and efficiently, in a dynamic computing environment. The importance of this software which stands between the user workloads and the platform or the applications and the resources, is undoubtful. The continuous hardware evolution proposing new challenges for the management of resources and the complex specifications of highly compute-intensive applications implying additional parameters for jobs scheduling, made the function of a Resource and Job Management System more complicated than ever and turned this intermediate software into a multi-facet research tool.

1.1 Motivation

The research on High Performance Computing (HPC) is defined by challenges on all different levels of computer science. Hardware, middleware and application layers have presented very significant advances in the last few years. Naturally, the evolution begun on hardware with new multiprocessing architectures, graphics accelerators and high-speed networks, which became the new components of the modern HPC infrastructures. On the other side, the applications had to adapt themselves with programming languages evolution, for parallelized (MPI) and multithreaded (OpenMP) codes in order to acquire the most of the computing hardware. The Resource and Job Management system stands between those two different layers and its role is to efficiently exploit the capabilities of the hardware by fairly distributing computing power to applications and provide benefit to the whole community.

A lot of research issues have arised in the area of scheduling and management of resources and jobs, so as to cope up with the latest evolutions on hardware and applications layers. Inspired by different advances of technology and in order to answer to particular application needs, scientists have constructed various kinds of computing platforms to perform High Performance Computing. Clusters, multi-clusters, grids, desktop or lightweight grids and lately clouds propose different ways for executing workloads and applications. Even if additional middleware software may be needed, the main middleware component of the above platforms is still the local RJMS. Hence interfacing capabilities for the support of differnt HPC environments have to be take into account.

Taking into consideration the current technological evolutions, we want to imagine a system that will be capable to achieve an efficient exploitation of all available resources. Studies upon various workloads of different systems [4] show that there exist many periods where the cluster is not fully utilized. This under-utilization periods of a system can be due to jobs traffic characteristics, like job interarrival times and cancellation rates. For example, systems that have big jobs interarrival times present low system utilization [5]. An overview of previous works and archived workload traces [4], [6] proved that most of the times, clusters are utilized less than 65% of their overall capacities throughout the year. Table 1.1 provides the utilization percentages of certain large scale parallel systems in production use. These workload logs are collected and maintained by Feitelson in [4]. The bigger utilization percentage 87,6% is observed on the LLNL Thunder cluster with 4008CPU Cores. It's interesting to see that the utilization percentage of the largest cluster on this table LLNL Atlas with 9216CPU Cores is only 64.1% throughout a period of 8 months.

The research that is made through this thesis lies on the following question: How can we take

Workload Traces	From	Until	Months	CPUS	Jobs	Users	Utilization %
LANL O2K	Nov 1999	Apr 2000	5	2,048	121,989	337	64.0
OSC Cluster	Jan 2000	Nov 2001	22	57	80,714	254	43.1
SDSC BLUE	Apr 2000	Jan 2003	32	1,152	250,440	468	76.2
HPC2N	Jul 2002	Jan 2006	42	240	527,371	258	72.0
DAS2 fs0	Jan 2003	Jan 2004	12	144	225,711	102	14.9
DAS2 fs1	Jan 2003	Dec 2003	12	64	40,315	36	12.1
DAS2 fs2	Jan 2003	Dec 2003	12	64	66,429	52	19.5
DAS2 fs3	Jan 2003	Dec 2003	12	64	66,737	64	10.7
DAS2 fs4	Feb 2003	Dec 2003	11	64	33,795	40	14.5
SDSC DataStar	Mar 2004	Apr 2005	13	1,664	96,089	460	62.8
LPC EGEE	Aug 2004	May 2005	9	140	244,821	57	20.8
LLNL uBGL	Nov 2006	Jun 2007	7	2,048	112,611	62	56.1
LLNL Atlas	Nov 2006	Jun 2007	8	9,216	60,332	132	64.1
LLNL Thunder	Jan 2007	Jun 2007	5	4,008	128,662	283	87.6

Table 1.1: Logs of Real Parallel Workloads from Production Systems[3]

advantage of these unutilized periods of the infrastructures by efficiently exploiting the otherwise idle resources and turn this use into beneficial values towards the platforms and the users?

1.2 Research Subjects

The main areas of our research are focused on the field of Resource and Job Managment Systems. This dissertation presents a deep analysis of the concepts, structure, architectural design and functionalities of those systems. We have closely followed their adaptation to technological evolutions and modern applications needs and we analyze all involved research issues like efficiency and scalability. Our principal concern lies on the improvement of the Resource and Job Management System. Hence, we present studies to provide efficient exploitation of the infrastructures, by utilizing otherwise idle resources, without neglecting quality of services to the users. In order to be able to make our studies upon this field of computer science, we have developed an experimental methodology based upon deployment of real-scale platforms and evaluation techniques through workload traces replay.

1.2.1 Scheduling and Management of Resources and Jobs

A Cluster Resource and Job Management System (RJMS) is responsible for the efficient assignment of users jobs upon the cluster's computational nodes. The work of the RJMS is a multi-facet

process. It involves optimal scheduling procedures among the users' jobs, efficient matching of those jobs upon the computing resources and consistent management of resources for sequential and parallel jobs launching. Due to the latest technological hardware advances (multicore architectures, high-speed networks, etc), system components have become more complicated with deeper hierarchy layers. This implies the use of finer resources' management techniques from the side of the middleware (RJMS). On the other hand, the high complexity of users' applications (multithreaded, highly parallel, hybrid, etc) struggling for computing power, need an efficient and robust system that provides high-performance execution and quality of services.

The rapid growth of clusters' computing resources and the entrance to petaflop scale for supercomputing production clusters [3], have created the needs for further research in the area of Scheduling and Management of Resources and Jobs. The efficient assignment of large number of resources coresponding to a large number of users produces issues like job launcher's and scheduler's scalability. Propagation and scheduling algorithms need to be sufficiently equipped to deal with these additional complexities. Moreover, the increase of internal node processor hiearchies (multicore and manycore) along with the growth of network diameters, demand methods for efficient placement of parallel and multithreaded jobs upon particular groups of nodes and cores which can provide optimal communication speeds. The big energy demands of the large clusters have raised the issue of energy consumption [7]. All these mechanisms need to be taken into account during the RJMS scheduling process. More hardware components mean larger propabilities for failures. Hence, the high availability of the RJMS has to be to be treated as well.

Our research explores fundamental concepts of Resource and Job Management Systems. We present a thorough comparison of some of the most known RJMS and we evaluate their performance upon controlled platforms and under realistic workload conditions.

1.2.2 Techniques for efficient resources exploitation

The under-utilization periods that the platforms are submerged can be efficiently exploited by the systems middleware where the local Resource and Job Management System (RJMS) has the main role. But even if we consider that the middleware (RJMS) is sufficiently equipped to detect those periods, not all applications will be able to adapt themselves, on-the-fly, to the new resources that may become available. So we will need specific kind of applications that will be capable to resize themselves on platforms' availabilities.

Optimizing system utilization with checkpoint/restart mechanisms

The bag-of-tasks applications are sequential tasks with no dependence amongst them . Since,

there is no need for task-synchronization, during the computation, the application can adapt itself upon the available resources. This kind of applications represents one of the mostly used classes of scientific applications today [8]. The aggregation of otherwise idle resources for execution of bag-of-tasks applications is a widely used trend [9], [10], [11], [12], especially in the context of grid computing. Driven by the reputation and the wide utilization of this kind of platforms, we initially place our research in a similar context. More precisely we focus upon a lightweight grid configuration where clusters share common administrative choices. The applications are provided in the form of bag-of-tasks and the jobs are executed upon clusters as *best-effort* tasks. The former, are lower priority tasks executed upon otherwise idle resources, that get instantly killed whenever normal jobs need the resources. In a system like this, the large number of interference failures provide an additional motivation for fault-tolerance.

Efficient techniques dealing with failures may provide optimization for the system exploitation. Checkpoint/Restart is a common technique for tolerating failures and optimizing jobs turnaround times. Applications' state is periodically saved to reliable storage, and in case of failure, restarts from a prior state. In our study, we consider an application transparent, system level checkpoint/restart technique that would be a solution to guarantee reliability for all kinds of applications.

Under this context, we provide a method to enhance the default scheduling mechanisms of bag-of-tasks applications with checkpoint/restart mechanisms, in order to achieve faster jobs' turnaround times and better overall system utilization (chapter 4).

Improving resources' exploitation with Malleability

Our second area of research is centered upon the subject of malleable tasks scheduling. The dynamicity of these applications is triggered by the platforms availabilities: a malleable application can *grow* whenever resources are available and can *shrink* when they are requested by the resource and job management system [13]. The scheduling community has been especially interested on this kind of applications because their high adaptation capabilities can improve both systems utilization and jobs' response time [14], [13]. Whereas, a lot of progress has been made in theory, in practice the support of malleable tasks upon a RJMS is a very difficult procedure due to various complexities upon different levels [15]. Issues concerning programming models, scheduling policies and communication protocols between applications and the RJMS need to be seriously considered. Advancing the study of this last field around the Resource and Job Management Systems, we design an approach for supporting malleable applications upon a 'versatile' and easy to extend RJMS. We have implemented a simple communication protocol to allow interaction between the RJMS and the applications and we developed techniques for scheduling a malleable

application without changing the core scheduler of the RJMS.

Unfortunately due to the complexity of their programming model, malleable applications are difficult to program and the bigger class of parallel applications today are implemented as rigid ones [13]. Rigid applications cannot resize themselves during runtime and is impossible for them to efficiently utilize otherwise idle resources. In an effort to allow a wider spectrum of applications to take advantage of an efficient exploitation of the system by using otherwise idle resources, we went one step further. We have designed and implemented a trasparent technique to provide malleability for rigid applications, upon multicore architectures, through the RJMS. We take advantage of specific mechanisms of multicore CPUs along with linux kernel advances for cpu affinity and we provide transparent expanding and folding of rigid parallel applications.

Finally, we analyse the trade offs between gains using these two different forms of Malleability versus a simple moldable approach for exploitation of otherwise unutilized resources and we study the complexities of the malleability implementation upon a Resource and Job Management System.(chapter 5).

Energy-efficient resources' exploitation

The use of otherwise idle resources for additional computations, gives faster scientific results. On the other hand, idle resources could contribute in reducing the systems' overall energy consumption. Indeed, during the last few years a new area of research, around energy efficiency, emerged as an imminent neccesity. As platforms size increase rapidly, their energy needs become enormous [16]. A way to treat this growing problem is to improve the energy efficiency at different abstraction layers. The evolution on the hardware level helped for the construction of new low-power High Perfromance Computing Systems [17]. Systems like Green Destiny [18] addressed these problems by significantly reducing per-node power consumption. Even if the hardware components are directed towards reduced energy consumption, the middleware and applications need to adapt themseleves to face the new chalenges.

According to prior work, idle computing resources consume a very important amount of energy [19],[20]. This energy consumption can be avoided if specific actions take place. Under this context we extend our research in order to take advantage of otherwise idle resources and achieve energy-efficient system exploitation. More specifically, we have implemented a resource management optimization upon an extensible RJMS so as to power-off, otherwise idle computing cluster nodes, under specific conditions. Following this method a system can benefit of its idle periods and perform energy reduction. As far as the application level is concerned, many efforts have been made in order to provide energy-efficient MPI programming [21] or special Dynamic Voltage

and frequency scaling mechanisms, for reducing energy on MPI programs [22]. Following those ideas, we developed specific options upon the RJMS that support CPU frequency scaling and Hard disk spin-down upon modern platforms (that provide this kind of hardware treatment) and allow applications to take advantage of these features.

So under the same context of using the otherwise idle resources for better system utilization, we propose energy-aware system management techniques (chapter 6).

1.2.3 Experimental Methodology based on workload traces

The Resource and Job Management Systems (RJMS) for High Performance Computing have a complex behaviour, defined by various functionalities and parameters. A lot of research has been made upon methods for evaluation of specific layers of the RJMS stack, like the scheduling [23],[24]. Most of the times, those methods are based upon simulation which can provide reproducable experiments with valuable results. However, it focuses only on a specific behaviour or mechanism, abstracting the operation of the rest of the system. To be able to evaluate the behaviour of the RJMS as a complete unit, we use real experimentation upon dedicated, controlled platforms.

We have developed a methodology based on real-scale experimentation that allows to evaluate the complex behaviour of the RJMS, taking into account all different parameters and internal mechanisms like scheduling, job launching, network topology placement, even energy consumption. To provide realistic conditions and interesting observations for our experiments, not only the dedicated real-scale platform but other parameters also are important to be defined. The most critical of those are the system factors, users workloads, applications to be executed and evaluation metrics. Based upon the research upon scheduling systems evaluation through simulation, we adopt their techniques with various workload executions and we propose the replay of synthetic or real production workload traces.

This method allows to evaluate different mechanisms or functionalities of the RJMS under realistic conditions. Using this methodology we experiment with the various optimizations proposed during our study and perform comparisons between different Resource and Job Management Systems.

1.3 Thesis Overview

The organization of this thesis is as follows: In chapter 2 we provide the experimentation methodology that we use to perform the experimental validation of our implementations and the performance evaluations. Chapter 3 presents Background Information and the state of the art upon Resource Management and Job Scheduling for High Performance Computing, presenting conceptual and experimental comparisons between known Resource and Job Management Systems. The following three chapters present the main part of our contributions centered on proposed techniques for efficient resources exploitation. Chapter 4 describes a method for efficient system exploitation through a checkpoint/restart mechanism. This chapter is based on material published in [25, 26]. Chapter 5 presents the support of malleability upon a Resource and Job Management System. This work has lead to the following publications [27, 28]. Chapter 6 describes the techniques used to provide energy consumption reduction upon clusters through proposed functionalities upon a Resource Manager. The work in this chapter is based on material published in [29, 30, 31]. Conclusions and future work perspectives, are finally presented in chapter 7.

Chapter 2

Evaluation Methodology for Resource and Job Management Systems

Research in Clusters, Grids and High Performance Computing is based on a variety of methodologies and tools. Technological evolution and scientific needs have made systems and applications more complex throughout the years and the study of the complete computing systems now depends on thousand of parameters and conditions. A large part of the research conducted in this field is mostly performed using simulators or emulators. These tools present advantages related to the control of experiments and the ease of reproduction but also limitations because they fail to capture all the dynamic, variety and complexity of real life conditions.

The area of Resource and Job Management for HPC implicates various procedures and internal mechanisms making their behaviour complicated and difficult to model and study. Indeed, mechanisms that may take place after a simple job launching upon the RJMS can include admission rules validation, scheduling, command propagation, tasks placement according to topology constraints, faut-tolerance even energy efficient management. In addition, every different mechanism depends on a big number of parameters that may present dependencies amongst them. Studying tradeoffs like energy efficiency versus system utilization, or application performance versus scheduling complexities, are only some examples that show how important it is to be able to study the system as a whole under real life conditions. Hence, even if simulation and emulation can provide important initial insights, the need for real-scale experimentation seemed undouptful for the study and evaluation of all internal functions as one complete system.

The real-scale experimental methodology presented in this chapter is based upon Grid5000

platform which is a testbed composed of a collection of various clusters and software tools dedicated to research on computer science. One of the main drawbacks of real-scale experimentation versus simulation and emulation is the difficulty of reconfiguration and reproduction of experiments. Grid5000 platform has been designed exactly to answer this problem, hence providing an ideal tool for large-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 and evaluate those systems and experiment with new optimizations upon particular RJMS, we need to define specific workloads and application profiles that are going to be used. Nevertheless, in order to choose the right workoads and applications, we are facing many choices and issues to deal with, depending on the type of experiment. In addition, the requirement to work on a real scale raises the need for development of particular tools for treatment and replay of those workloads under real-life conditions.

In the remainder of this chapter, we initially provide the background and related work upon real-life experimentation, workload characterization, workload modeling and benchmarking for parallel computing systems. The following section makes an analysis of Grid5000 platform and its design concepts, presents the experimentation procedure and discusses advantages and difficulties. The third section presents the actual evaluation procedure and describes the assumptions and the definition of parametrized system factors, the different approaches used for replay of workload traces along with the collection of tools that has been developed to analyse and replay the workload traces. Moreover, it provides the list of applications used for execution during our experimentation and the evaluation metrics used. Finally, section 4 concludes and gives ideas for future work.

2.1 Related Work

Parallel and distributed computing is a widely active research area. In order to conduct research upon this domain the community has adopted various techniques based upon theoretical or experimental analysis. The mathematical approaches upon this area can turn easily into deadend paths, leaving experimentations to be a safest way for proof of concepts. Indeed, the complexity of the underlying platforms along with the big number of parameters that define the applications may result into unrealistic theoretical assumptions. Hence, most research results are obtained via experimental evaluation. Entering into details of Scheduling upon High Performance Computing systems research area, we also discuss upon the background and ongoing works of workload analysis, modelling and benchmarks for Resource and Job Management Systems.

Application	Environment	
	Real	Model
Real	In-situ	Emulation
Model	Benchmarking	Simulation

Table 2.1: Classification of experimentation methodologies [1]

2.1.1 Experimentation methodologies

In this section we present tools and testbeds that exist, for effectuating parallel and distributed computing experiments and we discuss upon their advantages and drawbacks. Gustedt et al. present a complete survey [1] of experimentation methodologies for large-scale systems. They classify experimental methodologies in 4 different classes depending on the kind of environment the experiment is made upon and depending on the kind of application it is being used 2.1.

The survey classifies the different existing methodologies according to criteria like: control of experiments, reproduction ability, abstraction level, scalability for large-scale experimentation, execution speed and speed-up between an experiment and a real execution, processor folding and heterogeneity management of the platform.

Considering their analysis and the related work on experimental methodologies for parallel and distributed computing, we conclude on the following remarks upon the different methods:

- **Simulation**

  Simulation focuses on a specific behavior or mechanism of the distributed system and abstracts the rest of the system. The main advantage of simulation is that it allows highly controllable and reproducible experiments with a large set of platforms characteristics and experimental conditions. The design of a simulator requires a lot of modeling work and provides a very difficult scientific challenge. Simulation is the most widely used methodology for experimentation in Computer Science. However, not all factors and conditions that influence the distributed system can be fully experimented and only a model of the application is executed and not the application itself. A lot of simulators exist that model CPU at the hardware level [32]. In networking, the most famous tool is the Network Simulator (ns2) [33], which implements a very large collection of protocols and queuing models and grounds most of the research literature on the field. The emerging tools used in Peer-to-Peer community encompass PlanetSim [34] and PeerSim [35]. Their main goal is scalability (from

hundreds of thousands of nodes, up to millions of nodes), at the price of a very high level of abstraction. Both GridSim [36] and SimGrid [37] aim at simulating a parallel application on a distributed grid environment. GridSim is mainly focused on grid economy studies, while SimGrids authors claim a larger application field to any application distributed at large scale. Scalability comparisons show the clear advantage of SimGrid over GridSim to that extent.

- **Emulation**

  Emulation is the methodology which provides a tool capable for executing the actual software of the distributed system, in its whole complexity, using only a model of the environment. The challenge here is to build a model of a system, realistic enough to minimize abstraction, but simple enough to be easily managed. ModelNet [38] is a tool that is principally designed to emulate the network components. It works at real-scale under a real hardware. Network packets are routed through core ModelNet nodes which cooperate to limit the traffic in terms of the bandwidth, of congestion constraints, of latency, and of packet loss. This tool only models the network, ignoring the CPU and disk resources and is not very scalable. Emulab [39] shares similar goals with ModelNet and provides network emulation. Microgrid [40] allows to emulate the parallel execution of distributed application using just a few (possibly one) processor(s). It emulates both the CPU and the network performance. Each, emulated resource executes, in a confined way, part of the applications. The approach of Microgrid is based upon virtualization technology as it allows several guests to be executed on the same physical resource. Even if Microgrid is able to emulate tens of thousands resources, it is unfortunately not supported anymore and does not work with the recent compiler versions such as gcc 4. In order to tackle the problem of emulating an heterogeneous environment Canon et. al. have designed Wrekavoc [41] system. The main objective is to have a configurable environment that allows for reproducible experiments on large set of configurations using real applications. Wrekavoc degrades the performance of nodes and network links independently in order to build a new heterogeneous cluster. Then, any application can be run on this new cluster without modifications. However, contrary to Microgrid, Wrekavoc needs one real CPU per emulated CPU. The CPU degradation is based on user-level process scheduling (processes are suspended when they have used more than the required fraction of the CPU).

- **Experimental Platforms**

The Production platforms could provide a good solution for real-life experimentation. However, the fact that specific experiments need specialized software which is often hard to install on production platforms, along with the big difficulty of experiment reproduction, are some important limitations. That's why scientists have designed and developed experimental platforms to provide better control and reproduction abilities for real experimentations. Many institutes and international programs have developed various tools to foster large-scale distributed systems research. Platforms like PlanetLab [42], GENI [43], DAS [44] and Grid5000 [45] provide some significant examples.

The purpose of Grid5000 is to serve as an experimental testbed for research in grid computing and large-scale systems. It aims at building a highly reconfigurable, controllable and monitorable experimental Grid platform, gathering nine sites geographically distributed in France plus one in Luxembourg and one in Porto Alegre, Brazil. PlanetLab is an international-wide testbed. It is based upon virtualization technologies in order to allow different experiments, using different but possibly overlapping set of nodes concurrently. The Dutch project DAS, which is a national-wide project has been designed to allow the reconfiguration of the optical network that interconnects the different clusters. The main difference of Grid5000 with all these platforms is the degree of reconfigurability. This functionality, allows researchers to deploy and install the exact software environment they need for their experiments, making the platform an ideal tool for real-life experimentation upon all layers of software stack.

The work presented in [46] provides analytical studies for validation of real-life experimentation methodologies. In our context, we follow the ideas presented on this article to provide the performance evaluation methodology for resource and job management systems.

During this thesis we have performed experimentations using only an experimental platform with synthetic or real applications. Nevertheless, we understand that the best methodology is to be able to perform both simulation and real experimentation. We argue that the ideal experimentation methodology should use simulation at a first place, based upon the easy control and reproduction, to obtain fast results for the validation of a prototype and in the same time observe specific metrics or trade-offs; The real experimentation should come at a second place, where taking into consideration the simulation results, we should perform experimentation with real conditions in order to be able to captuere all the complex behaviour and interaction between the different stacks of hardware and software and try to validate the initial observations in practice.

2.1.2 Workload modelling, characterization and benchmarks

Performance evaluation is effectuated by having the system's scheduler schedule a sequence of jobs. This sequence is the actual workload that will be injected to the system. A lot of research [47], [23], [24], [48], [49] has been effectuated upon workload characterization and modelling of parallel computing systems.

In order to model and log a workload of a parallel system, Chapin et al. [24] have defined the *standard workload format* (swf) which provides a standarized format for describing an execution of a sequence of jobs. The data fields of workloads following this format contain one line per job, that contains a list of space separated integers. Some of the most important included fields are: job number, submit time, wait time, run time, number of allocated processors, status, etc. An extension of swf format called *gwf*, has been proposed in [50], to characterize grid workloads. Some of the additional fields that have been introduced to reflect grid needs are: Requested type of architecture, Project id, network usage, disk space usage, etc. As we will also note during this thesis more fields were found neccessary to include and they are proposed to the above standards. For example a field to reflect the energy consumption of a particular job provides an important characteristic to modern architectures for compute and energy intensive applications.

There are two common ways to use a workload in order to evaluate a system.

1. Either use a *workload log*, also known as workload trace which is a record of resource usage data about a stream of parallel and sequential jobs that were submitted and run on a real parallel system,

2. or use a *workload model*, also known as synthetic workload, which is based on some probability distributions to generate workload data, with the goal of clarifying their underlying principles.

Both approaches have advantages and drawbacks. In any case, the main problem of both approaches is their degree of representation. That is, to what point do they reflect the actual conditions that the system will encounter in practice? The advantage of using a real-systems workload trace is that it reflects reality by keeping all the complexities met on the real workload, thus producing more reliable results [51]. Most production HPC systems maintain accounting logs for administrative purposes. These logs contain valuable information about all different activities upon the machine and in particular, about the attributes of each job that was executed (such as submission time, start time, end time, number of allocated processors, etc). A large archive of production HPC systems workload logs is maintained by Feitelson in the Parallel Workloads Archive [4].

Models always simplify reality, and using the wrong statistical model can yield misleading results. On the other hand, workload models [48], [52] give the advatntage of offering an absolute control on all features of the model simplifying the experimentation procedure, whereas the real workload trace reflects a specific system use and may depend upon specific system configuration, like the choice of clusters architecture. Many workload models exist in literature and most of them are described in [4]. The construction of a model is not an easy task and it is usually derived from some statistical analysis of real workload logs. An important parameter of a workload model is the distribution of the various workload attributes. Initial studies were using exponential (Poisson process) or uniform distribution for interarrival times, which later proved not representative enough. Other important factors that can characterize models are the correlation between different attributes such as job size, interrarival time and duration.

The model initially proposed in [53] and [52] was one of the first benchmarks to actually measure the performance of a real scheduler. However, the first version of this model (ESP-1) was constructed mainly to fill the needs of a specific system (NERSC) and was difficult to be adapted on other systems. Due to the unique characteristics of this benchmark, this research was submerged into significant critisism in order to be enhanced with improvements [54]. Based on these critics and in order to have an easily adapted benchmark, used for different scales and by different communities [55], a new model (ESP-2) has been proposed [56, 2] which provides the 2nd and current version of the ESP benchmark.

Kramer in his PhD thesis [2] proposed a framework for a user based methodology to evaluate high performance computing (HPC) systems. In more detail, this methodology (PERCU) explored five components that define an HPC system: *Performance*, *Effectiveness*, *Reliability*, *Consistency*, *Usability* and introduces a set of new measurement methods along with improvement approaches for each different component. The proposed measurement method for *Effectiveness* was ESP benchmark and an in depth analysis of ESP-1 and ESP-2 tests were made in [2]. Moreover, various use cases were presented where the ESP utilization prooved to be beneficial not only for computing systems but for resource managers as well. ESP-2 benchmark has been adopted and included into our methodology for Resource and Job Management systems evaluation and for the sake of simplicity, when we mention ESP benchmark we will always refer to this second version ESP-2 of the test.

We assume that the observed performance results will reflect important differences between systems or interesting conclusions for the optimization tecnhinques. However, as stated by Feitelson on [57], performance differences can be also an artifact of the evaluation methodology. So

we need to study not only the factors that affect system behavior, but also those that affect the evaluation procedure and hence the measures. The actual definition of the evaluation methodology will play an important role to the significance and validity of the final results. Some important comments that need to be taken into account in order to deal with the parallel job scheduling compromises and tradeoffs, are discussed in [49].

An information that is normally not included in the workload, is the type of application to be executed. Given the real-scale experimentation, the applications to be used can greatly affect performances and evaluations results. Depending on the specific feature under evaluation and its dependence on application performance, the methodology proposed in this study makes use of either: (i) real applications, (ii) parallel synthetic applications or (iii) simple sleep jobs. Furthermore, the experimentation methodology would be limited if we do not define in advance the evaluation metrics. Indeed, the definition of the proper evaluation metrics will indicate the experimentation needs and will help to direct the study of more important observations with reliable results

2.2 Real-scale experimentation upon Grid5000 platform

The complexity of Cluster and Grid systems have raised the need for real-scale experimental platforms where computer scientists run experiments, observe the distributed systems behavior at large scale under real-life conditions and make precise measurements.

As a matter of fact, the experiments upon complex distributed systems span over all the layers of the software stack between the user and the hardware. The applications, programming environment, runtime systems, middleware, 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.

Grid5000 was designed to answer to the above needs. It is a large-scale distributed platform that can be easily controlled, reconfigured and monitored. Especially designed for computer science, it provides a real-life experimental tool, ideal for research upon complex distributed systems.

2.2.1 Grid5000 Design and Architecture

During the preparation of the project in 2003, designers of Grid5000 conducted an analysis on the need of a computer science Grid and the diversity of potential experiments. As described thoroughly on [45], the analysis concluded the need for a large scale (several thousands of CPUs),

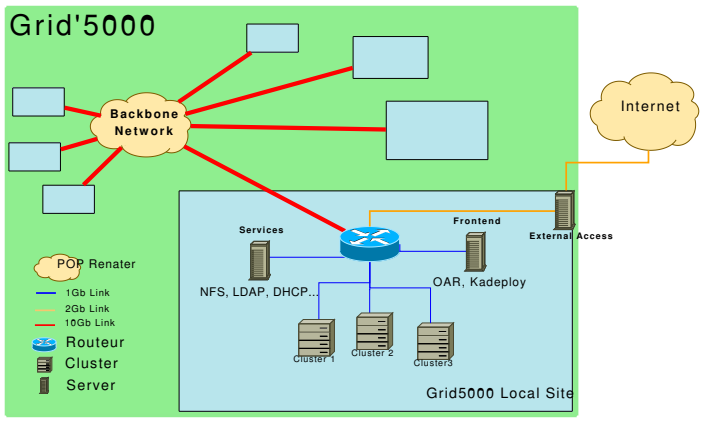


Figure 2.1: Overview of Grid5000 architecture

distributed (10 sites) computer science Grid. Moreover, the experimentation should cover all layers of the software stack, from the application layer to the networking protocols. Researchers may need a specific experiment setting, different from others. Those involved in networking protocols, operating systems and grid middleware often require a specific OS or kernel for their experiments. Their needs may be quite diverse when they are doing research upon grids: some require Globus, while others need Unicore or Boinc desktop grid middleware.

As a consequence, Grid5000 was designed to provide a deep reconfiguration mechanism allowing researchers to deploy, install, boot and run their specific software environments, possibly including all the layers of the software stack. This reconfiguration capability led to the experiment workflow followed by Grid5000 users: 1)reserve specific resources of Grid5000, 2)deploy a software environment on the reserved nodes, 3)run the experiment and make precise measurements, and finally 4)collect results and relieve the machines.

Since researchers are able to boot and run their specific software on Grid5000 sites and machines, the need of security mechanisms arose. Hence it was decided to isolate Grid5000 from the rest of the Internet but to let packets fly inside Grid5000 without limitation. The first choice guarantees that internet will be protected from the power of Grid5000 and the second one makes sure that communication performance will not suffer from the overhead of an imposed security system. Thus, Grid5000 is built as a large scale confined cluster of clusters. Strong authentication and authorization checks are done when users log in Grid5000 to prevent hacker attacks. In order to provide large hardware choices, Grid5000 aimed to use 1/3 heterogeneous and 2/3 homogeneous resources, so as to facilitate all kind of experiments.

As stated on [58] by Feitelson, the capability to reproduce experimental conditions is fundamental in computer science, especially when performance comparisons are conducted. To fulfill this strong requirement, we decided to use dedicated network links between sites, allow users to allocate dedicated nodes for their experiments and let them install and run their proper experimental condition injectors and measurements software. Thus every user has full control of the allocated Grid5000 partition.

Grid5000 initial goal was to provide 5000 CPUs distributed over different sites in France. This goal was reached and even over-passed having Grid5000 providing today more than 6500 CPUs distributed over 9 sites in France, 1 in Brazil and 1 in Luxembourg. Figure 2.1 presents an overview of the platform's architecture. Every site hosts a cluster and all sites are connected by high speed network links (RENATER 4: 10 Gbps links).

Every user has a single account on Grid5000. Every Grid5000 site manages its own user accounts and runs an LDAP server. On a given site, the local administrator can manage its user accounts. Once the account is created, the user can access any of the Grid5000 sites or services (monitoring tools, wiki, deployment, etc.). User data are kept local to every site and distribution to remote sites is done by the user through classical file transfer tools (rsync, scp, sftp, etc.). Data transfers from and to the outside of Grid5000 are restricted with secure tools and done through gateway servers.

At cluster level, users submit their resource reservations and experiment jobs using the OAR resource and job management system. To reconfigure the software stack on every reserved node, the users run the Kadeploy toolkit deploying the user defined software environment on a disk partition of selected nodes. Finally to monitor and control the experiments various software tools have been developed by Grid5000 scientists to facilitate different tasks of the experimentation process. In the following sections we present a detailed analysis of these key components used on Grid5000.

Cluster resource management and job scheduling: OAR

OAR [59, 60] is an open source Resource Management System for large clusters. Initially developed as a tool for research upon the area of Resource Management and Batch Scheduling, this software has evolved towards a particular flexibility. It provides a robust solution, used as a production system in various platforms like the regional grid infrastructure Ciment which is exploited by scientific computations in disciplines like environment, chemistry, astrophysics, etc. OAR is

the Resource and Job Management System selected to function on all Grid5000 sites. It is responsible for resources allocation and reservation along with the jobs execution. In OAR, resources required by jobs are matched with available ones. This matching is based on a hierarchical affinity of resources which enables them to allocate from a whole cluster until a specific CPU, Core or thread. Detailed description of OAR Resource and Job Management System will be given on the following chapter of this thesis.

In order to be able to execute experiments on different clusters of Grid5000, simultaneously, OARGRID [61] software has been developed. This toolkit is a wrapper that enables the use of several OAR clusters, of different sites, for easier grid experimentation. It provides the possibility of resources allocation and reservation on a grid level. The individual job execution is effectuated by the local OAR system.

Reconfiguration mechanism: Kadeploy

In the context of high performance computing research, scientists seem to need various software environments in order to perform their experiments. A software environment contains all the software layers like the operating system, the libraries, the middlewares and the applications (figure 2.2). According to their experiments' nature and the software layer they are investigating (protocols, OS, etc), they often require specific OS. Hence, a special tool with a deep reconfiguration mechanism allowing researchers to deploy, install, boot and run their specific software images, is needed.

Kadeploy [62, 63] is a software environment deployment tool designed to deal with the above issues, providing automated software installation and reconfiguration mechanisms on all the layers of the software stack. By using kadeploy, in a typical experiment sequence, a researcher reserves a partition of the cluster or grid, deploys its software image (figure 2.4), reboots all the machines of the partition, runs the experiment, collects results and finally relieves the machines. This reconfiguration capability allows researchers to run their experiments in the software environment that perfectly matches their needs and provides to users, a software homogeneous grid.

This tool uses the traditional protocols for network booting: PXE, TFTP, DHCP. As we see on figure 2.3, kadeploy architecture is designed around a database and a set of specialized operating components. The database is used to store all necessary information for the deployment process, the computing nodes and the environments. At the same time, the code is written in Perl, which is perfectly suited for system tasks. In addition uses a fast mechanism of environment diffusion which hardly depends on the number of nodes. This mechanism is based on a pipeline approach (chain of

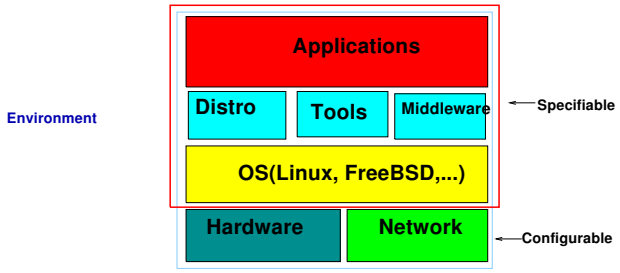


Figure 2.2: Software environment

TCP connections between nodes). This enables operations of deployment on large clusters (1000 nodes).

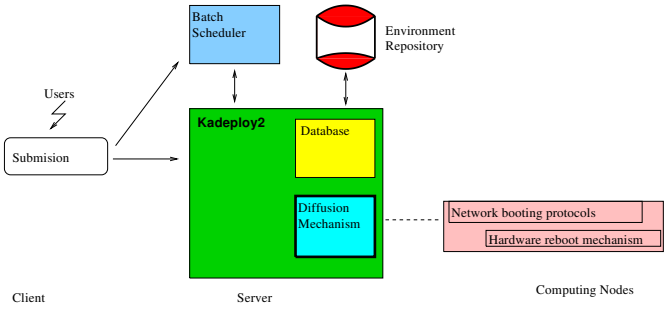


Figure 2.3: Kadeploy architecture

The deployment process 2.4 will write a complete environment, on a partition of the disk of each computing node, which will be followed by a reboot on this partition. The process ensures that the partition of the disk where the reference environment of the node is installed, remains intact during diffusion. To guarantee a greater function reliability, Kadeploy tool directs clusters to be coupled with remote mechanisms of hardware reboot. Thus, if a particular problem occurs on one or more nodes during a deployment, a restarting on the reference partition is ordered automatically, on defected nodes.

An environment is created very simply by making an archive of the root partition in compressed `tar` format. To ensure a high level of portability and to permit that an environment is usable on various clusters of similar processor architectures , the environment should not contain information corresponding to the initial cluster. That is possible because the majority of the services have

autoconfiguration mechanism (ex: protocol DHCP for the network) and the majority of the operating systems have hardware autodetection mechanisms making it possible to adapt to the minor differences (network cards, disks...). For the services that lack autoconfiguration procedure during the deployment, a procedure known as post-installation process supplements the parameter setting.

Grid5000 uses kadeploy environment deployment tool for effective reconfiguration capabilities.

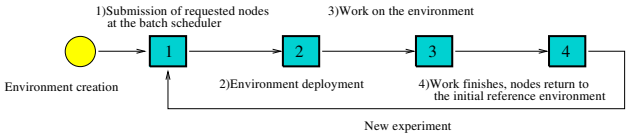


Figure 2.4: Typical sequence of an environment deployment.

2.2.2 Automatization and Reproduction techniques

Grid5000 testbed gave us a complete tool to perform large-scale experimentations under realistic conditions. We took advantage of all the provided features and toolkits of Grid5000 so as to facilitate our experiments. However, the issue that arised, was the ability to reproduce the experiments so as to validate our results. In order to answer to this issue we had to automate the whole procedure so as to be able to perform experiments without human intervention [64]. Hence our methodology is based upon a suite of Bash scripts that automates the whole procedure of experimentation. This procedure allowed us to deploy, install and configure a personalized HPC infrastructure (cluster or grid, according to our needs) in order to test our proposed optimizations or perform evaluation comparisons between different RJMS.

The process of the experimental methodology upon Grid5000 is composed by the following steps:

1. **Construction of the experimentation environment**, where the installation of needed operating system, kernel and software is performed. This step is normally effectuated only once, either in the begginning or whenever an environment update is needed.

2. **Selection and allocation of specific number of nodes** to be used for experimentation. This step implicates the use of OAR and OARgrid for allocation or advanced reservation of nodes.

3. **Environment deployment** upon the attributed computing nodes and **system configuration**. In this step we make use of kadeploy and specialized configuration scripts for configuration of the personalized cluster.

4. **Actual experiment execution and performance evaluation**. In this phase we need to extract and store all important information of the execution in order to be able to make a-posteriori observations. The experiment execution phase along with the definitions of the metrics that we will be using, is analysed with detail in section 3.

5. **Results collection**, and nodes relieve. Finally we collect all results and release the allocated nodes.

The reconfiguration mechanism, based on Kadeploy, allows the deployment of the constructed environment on the number of nodes that are requested for the experiments. Taking advantage of this capability a user can very easily deploy his own cluster or grid upon the Grid5000 platform. In more detail, for our experiments we constructed an environment containing all the needed software installed and ready for configuration. Once the environment is deployed on all the allocated nodes, we proceed to the software configuration and the attribution of the roles (Grid server, Cluster server, computing node, ...) just by choosing the specific services to be launched to each deployed node. Hence, we obtain a personal cluster or grid infrastructure ready for experimentation.

During this thesis all experimentations were effectuated upon Grid5000 following this same procedure.

2.3 Evaluation Methodology based upon Replay of Workloads

In the context of our study, the performance evaluation is used for Resource and Job Management Systems comparisons and for experimetation and validation of the optimization techniques for better system exploitation. The evaluation methodology upon real-scale configurations of High Performance Computing systems is a complex issue that implies close dependencies upon plenty of factors. The whole system is literally functioning under realistic conditions. Therefore, we can make changes upon all different kind of factors that influence the system, the execution and the final results.

System factors like scheduling policy, energy efficient resources management, task affinity or general system tuning can definitely change the performance results. In addition, the workload injected to the system has proved to be of great influence to the final results. How should we

know which workload trace to inject for a specific measurement? In our study we tried to answer to this question and proposed evaluations with both synthetic and real workload traces. Another important parameter is the type of executed applications. Whilst the users' workloads injected to the cluster we wonder what kind of applications should we execute during the allocation of resources. Depending on the specific measure that we want to obtain we propose a number of different applications (sleep, benchmarks or real) for execution with various profiles so as to stay very close to real computations. Finally the last (but not least) important factor of the evaluation methodology is defined by the actual metrics that we want to observe. What kind of parameter do we want to measure and what kind of results do we want to obtain? This is an important question that we had to pose to ourselves, in order to obtain valuable results.

In this section we analyzed the actual engine of the evaluation methodology into all the factors involved in the evaluation procedures during the various scenarios. Moreover, we present a collection of tools that we have developed, in order to analyze, visualize and replay workload traces so as to facilitate our experimentation.

2.3.1 System factors

The spectrum of parametrized system factors during real-scale experimentation can be particularly large, since it implies the tuning of both hardware and software parameters setting. The hardware choices are made during the selection of the right cluster for the experiments, according to specific hardware characteristics. For example the choice of the type of network: Infiniband, Myrinet or Ethernet will change the application performance and hence the final turnaround times of jobs. Usually we diminish the variations on hardware characteristics by selecting the same clusters for the same type of experiments, in order to obtain more fluidity in our results. Similarly, the OS, kernel and middlewares of the system can influnce evaluation performance and thus we inherit common software characteristics at least for the same type of experiments.

So, due to the complexity of real-scale experimentation we are obliged to assume that a lot of system factors will not be parametrized in order to be able to evaluate the factors and characteristics that really interest us. In the case of Resource and Job Management Systems comparisons, an important factor that influences overall performance, as we will observe on the next chapter, is the scheduling policy. This RJMS parameter defines the order that the jobs will be treated upon the clusters computing resources. Furthermore, every RJMS supports various policies, allowing to compare the efficiency of each one of them.

In the case of experimentation upon energy efficient exploitation, as it is discussed on chapter

6, the performance and energy consumption is measured for different scenarios that use normal or green resources management techniques. The green resource management techniques implicate the shut-down of unitilized machines for predetermined duration. The choice of normal or green resource management provides one parametrized factor. Furthermore, the variation of the predetermined duration related to machine unitilization, may result into different performances, and may be another interesting parametrized factor.

2.3.2 Workloads

Based on the related work in the area (section 2.1.2) it seems that to give precise, complete and valueable results the studies should include the replay of both modelled and real workload traces. In our studies we have developed the support of both ways of evaluation, however due to time constraints not all of our experimentations were capable to include the validation of both methods. Nevertheless, we believe that in those cases the reproduction of experiments and the variations of various factors results into reliable observations.

Evaluation based upon Real Platforms 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. The case of using a trace file of 32 dual-CPU nodes upon a 64 single-CPU nodes or a trace of 32 dual-CPU upon a cluster of 8 quad-CPU-dual-core node is possible but may present unreliable results, depending on the specific measures that we are willing to perform.

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 12 hours. This rule is one of the various usage rules enforced by Grid5000. However, the logs maintained in Parallel Workloads Archive 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 behaviour for observation?

As discussed on [57], underloaded systems present typically similar performances, whereas higher load conditions stress the system and expose differences in how systems react to load. Hence one of the first adopted characteristics that guide our selection, is the systems load. As it is explained on the following section a specific toolkit has been developed to calculate the system utilization of a workload log for specified duration windows. Therefore, various workload parts with different system loads may be used to test the system under different levels of stressing. Further, a more detailed selection observing the jobs interarrival and waiting times can take place with manual intervention. Big system load with long interarrival and small waiting times imply a simple behaviour with not a lot of stress: For example a case of only 2 jobs that use all the resources for a big amount of time might have the above characteristics and indeed there is not meaning of comaring the scheduling policies of an RJMS under a similar workload.

A specific group of workload logs that was widely used during this thesis was the one of DAS2 multicluster supercomputer. The main reason was the relative small number of nodes that defined the system from which the workloads were extracted along with the interesting variation of jobs characteristics. On [5] the authors present a comprehensive characterization of the DAS2 workload. The traces obtained, are written on the standard workload format [65] (swf) and represent twelve months of load.

Evaluation based upon Synthetic Workload

Concerning the synthetic workloads, our main interest was to use a validated model that provides easy adaptation upon any size of cluster in order to facilitate our already complex real-scale experimentation procedure.

To evaluate the scheduling performance of Resource and Job Management Systems, we have adopted the Effective System Performance (ESP) model [2, 56]. This is a widely used model, which not only provides a synthetic workload but proposes a specific type of benchmark application that can be executed to occupy resources as simply as possible, without stressing the hardware.

The ESP [2] test was designed to provide a quantitative evaluation of launching and scheduling parameters of a resource and job management system. ESP benchmark is a composite measure that can evaluate the system via a single metric, which is the smallest elapsed execution time of a representative workload. Multiple ESP tests may be performed in order to adjust the various RJMS parameters that can influence the exectuion of the workload (like scheduling policy, or propagation algorithms) and to tune the system for optimal results.

As presented in [2], the overall design goals and choices for the Effective System Performance

benchmark can be deduced to the following:

1. Complete independence from the hardware performance (like CPU speed) or compiler improvements on the executed application codes. This is addressed by proposing a simple MPI application with target run-time that can be fixed to a given value.

2. Ability for efficiency evaluation of different scheduling policies (like backfilling,preemption,etc) supported by the resource and job management systems. This is possible through the use of the same particular workload.

3. Ability for scalability evaluation of job scheduling and launching procedures. This is addressed by proposing a dynamically adjusted proportional job mix in order to use the benchmark and provide evaluations upon the same systems of different scales.

4. Capability of repetitions in order to evaluate the RJMS through its improvements over time.

The ESP test has been deliberately constructed to be processor-speed independent with low contention for shared resources (e.g. the file system) and a specific measure of scalability, stability and effectiveness of a system's RJMS. It runs a fixed number of parallel jobs through a manager of resources and jobs. Individually, the jobs have their elapsed run times set to a fixed target run time. The elapsed time of the total test is independent of the different hardware performance and is determined, to a large degree, by the efficiency of the scheduler and the overhead of launching parallel jobs. In ESP, there are 230 jobs derived from a list of 14 job types, which can be adjusted in a different proportional job mix, if needed. The test is stabilized to the number of cores by scaling the size of each job with the entire system size. Table 2.2 shows the job types with their relative size compared to the entire system, instance count and target run time. The ESP test includes two full configuration jobs, called Z-jobs in the test scripts, which are constructed to use the total number of available computational cores. The default ESP execution rules specify that the full configuration jobs cannot run at the beginning or the end of the test period and that no other job is permitted to start running in the interim between the submission of the Z job and its launch. In our case we have slightly simplified this last rule. This was made because only systems that allow preemption or checkpoint/restart would be able to respect this rule and our goal is to be able to use this method for a large spectrum of RJMS. There are two versions of the test the 'throughput' variant without the Z-jobs (228 jobs in total) and the 'multimode' variant where the Z-jobs are submitted at 2400 and 7200 seconds after the start of the test and after all other jobs A, ..., M have been submitted. In our experimentation we use only the 'multimode' variant and we give a higher priority to Z-jobs

Job Type	Fraction of Job Size relative to total system size	Count of the number of job instance	Target Run Time (Seconds)
A	0.03125	75	267
B	0.06250	9	322
C	0.50000	3	534
D	0.25000	3	616
E	0.50000	3	315
F	0.06250	9	1846
G	0.12500	6	1334
H	0.15820	6	1067
I	0.03125	24	1432
J	0.06250	24	725
K	0.09570	15	487
L	0.12500	36	366
M	0.25000	15	187
Z	1.00000	2	100
Total		230	

Table 2.2: ESP benchmarks synthetic workload characteristics [2] .

than the rest of them and let the system decide how these are going to be treated depending on its capabilities and supported parameters. The rules dictate that the first full configuration job is only submitted after a part of the workload has already been scheduled and is running. The jobs in the ESP suite is submitted to the RJMS in a pseudo-random order following a gaussian model of submission. They are separated into two blocks. The first block contains all jobs except the two job type Z jobs. After 40 minutes that the last normal job has been sent, the first Z-job is submitted, and after 2 hours the second Z job is submitted. No manual intervention is permitted once the test has been initiated.

The fractional-size is the size of the job as a fraction of total system size. For example, if the system under test has 10240 cores for computation, then the size of job-type F is 640 (= 0.06250 x 10240) cores. The ESP test can be applied to any system size and has been verified on 64, 512, 1024 2048, 6726 and 19,320 computational core systems. The specific value choices of job sizes fraction along with the jobs target run times were inspired according to the real workloads of a specific computing center [52]. The scientists wanted to construct a synthetic workload that will reflect the characteristics of their everyday workloads in order to make a better selection of the resource management software or the specific scheduling policy they needed to use for improvement of

their system utilization. The initial version of the benchmark contained even specific applications from different branches of science to execute [52], whereas the second and current version ESP2 [2] proposed a slightly different model and the execution of a simple toy MPI application (section 3.4.3).

Given a total amount of work, a hypothetical absolute minimum time (T-BEST) can be computed by dividing the work by the system size. Regardless the system size this hypothetical absolute minimum time is T-BEST = 10,773 seconds (3 hours). T-BEST is independent of the total system size and the processing speed of the system. The Effectiveness ratio, as show in [2], is the time the test actually runs compared to the time the best packing solution indicates, which means T-BEST divided by the observed elapsed time of the ESP test as shown on 2.1, where `x` is the concurrency for test code `i` and `T` is the run time the code `i` runs.

$$E = \frac{T - BEST}{\sum_{i=1}^{I} x_i * T_i} \tag{2.1}$$

`E` is the metric of the ESP test which allows us to make performance evaluations of a singe RJMS by varying some of their parameters and perform comparisons between different RJMS. For increasingly efficient systems, the ratio approaches unity. The T-BEST is simply a convenient definition of a lower bound. It is not possible to obtain T-BEST in a real test even in the optimal case. Therefore, most attainable ESP ratios fall in the range of 0.6 - 0.9.

In order to use it on our methodology we have extended ESP so as to support more resource and job management systems. Hence, the latest version of OAR, SLURM and Torque+Maui RJMS have been supported upon ESP.

Modifiying ESP for evaluation of topology aware placement techniques

The ESP benchmark can be used to compare systems of different scales, regardless the hardware characteristics. But in some cases we want to compare RJMS features (like topology aware placement techniques) using the same system and conditions. In this case the actual application performance and turnaround times of single jobs will provide variations on the final elapsed time of the whole workload. Limited by the fixed target run times of ESP model; the normal ESP benchmark does not allow us to evaluate features like these. Nevertheless, if we execute real applications with real (not-fixed) runtimes then different task topology placement techniques will lead to different application execution speeds, single jobs turnaround times and different whole workload elapsed

time. This will finally reveal insights upon the efficiency of the topology aware placement techniques. This is because applications that are sensitive on communications will perform better and have faster turnaround times if placed upon resources that allow faster communication and this will show on the single jobs execution times which will lead to smaller ESP elapsed times. In particular, in our case, we substitute the use of the simple default MPI application of ESP (section 2.3.3) with a NAS MPI application (section 2.3.3) which is sensitive to communications. More details about the particularly adapted ESP benchmark called *TOPO-ESP-NAS* are given in the description of the actual experimentation in section 3.5.3.

2.3.3 Executed Applications

The users workloads presented on previous section define the attributes of the jobs that are going to be injected to the system. Since we are dealing with real-scale experimentation, the actual definition of the applications to be executed needs to be made as well. Depending on the specific factors that we are willing to measure for every different type of experiment we need to choose the right application profiles. For example, in case we are only interested for the actual scheduling of all the jobs and no specific application performance for single jobs then simple sleep jobs, can be used, that only provide a time allocation for the RJMS and do not stress the system. On the other hand, if we need to take into account the overheads conducted by stressed network, machines CPU usage, or memory then synthetic or real applications can be used.

During our experimentation presented on the following chapters we have been using various types of applications.

Sleep applications

The definition of sleep applications consists of using the simple unix command `sleep` followed by a number which represents the amount of time in seconds that a Core, CPU or Machine will stays idle. The main advantage of using sleep jobs is that they represent the simplest type of application with a predefined steady duration that can be performed in any number of nodes. However, due to their simplicity we can use them only for experiments that the final evaluation measures are not influenced by CPU, bandwidth, or memory stress, or generally when we don't care about computations overheads.

Synthetic applications

Synthetic applications represent a specific type of applications defined by characteristics which are based upon real applications profiles. They can be used for computations with a goal to stress specific parts of the system like CPU, memory, network or even I/O. Furthermore, the fact that they can be parametrized according to the users experimental needs makes them ideal for real-scale experimentation. Their drawback is that even if they implicate real computations they cannot capture the complexity of a real application. The synthetic applications used for the experimentations in this thesis are parallel versions programmed with MPI message passing implementation and were selected to be the following:

- **NAS NPB3.3 benchmarks** The NAS parallel benchmarks [66], are widely used to evaluate the performance of parallel supercomputers. They exhibit mostly fine-grain exploitable parallelism and are almost all iterative, requiring multiple data exchanges between processes within each iteration. They consist of eight problems (five kernel and three simulated CFD applications) derived from important classes of aerophysics applications. The Benchmarks BT (Block Tri-diagonal), SP (Scalar Penta-diagonal) and LU (Lower-Upper Symmetric Gauss-Seidel) solve a discretized versions of the unsteady, compressible Navier-Stokes equations in three spatial dimensions.

  The NPB-3.3 MPI implementations of the benchmarks have good performance on multiple platforms and are considered as a reference implementation [1].

  Jaspal Subhlok et al. in [67] have made a valuable characterization of all different NAS benchmarks and we use this as a reference for selecting the right NAS benchmark for each experimentation according our needs for CPU, network or memory stress.

- **Linpack-HPL**

  LINPACK is a software for performing numerical linear algebra on digital computers. Initially develloped by Jack Dongarra [68] was intended for use on supercomputers. The LINPACK Benchmarks are a measure of a system's floating point computing power. They measure how fast a computer solves a dense N by N system of linear equations Ax = b, which is a common task in engineering. The solution is obtained by Gaussian elimination with partial pivoting. The result is reported in millions of floating point operations per second

[1] $http://www.nas.nasa.gov/Resources/Software/npb\_changes.html$

(MFLOP/s). For large-scale distributed-memory systems, HPL [69], a portable implementation of the High-Performance LINPACK Benchmark, is used as a performance measure for ranking supercomputers in the TOP500 list of the world's fastest computers.

It can be thought of as a test of many functions :

- cpu speed and instruction sets
- memory capacity
- system bus speeds
- interconnect topologies, performance and design
- linear algebra library optimizations
- compiler optimizations
- communication stack and protocol optimizations

The Linpack HPL benchmark contains several tuneable parameters, but its most important drawback is that it does not stress the I/O disk accesses. [2]

- **pchksum benchmark**

  This toy application has been implemented to be used and fit the needs for ESP benchmark [56, 2]. It is a simple parallel application that performs MPI communications. Each task generates a random message string and computes a digest. Each tasks sends the random message and receives a similar message from the next one. The rank of tasks are randomly generated. With each received message, the task does a XOR operation onto a message buffer which initially contained its random message. This operation is continued until a predefined requested time. The next part, repeats the above operation but in reverse order of the tasks ranks. After the last XOR operation, the updated message should be exactly the same as the initial message. Or, equivalently their digests should match. If this is met then the job has been correctly executed.

- **Mandelbrot set**

  This is a widely known exercise for parallel computation because its sequential algorithm can be simply programmed in a parallel version using MPI. In our context this application is used for a dynamic process management technique where the application is able to grow or shrink,

[2] $http://www.netlib.org/benchmark/hpl/tuning.html$

during the execution time, according to the resources availability. For the sake of some experiments, a static version of Mandelbrot was implemented to a dynamic version through lib-DynamicMPI with MPI_comm_spawn primitives. The dynamic Mandelbrot starts with a master that will spawn workers, manage tasks and store the results. Workers receive tasks, execute them, return results and wait for more tasks. The application is able to identify resources changes and launch the dynamic operations. In growing, new workers are spawned upon resources that become available and in shrinking workers inform the master which are the executing tasks and then finalize their execution.

Real applications

The advantage of real applications is that in any way it is the most representative method of testing the system especially if the application is widely used upon the platform you are willing to experiment it on. However, the recuperation of a real application and the opportunity of performing evaluations, that can allow you controlled reproducibility, with it, is really difficult to find and rare. During the research for this thesis we had the opportunity to use one real astro-physical application for the experimentations of one of our studies.

- **MCFOST astro-physical application** is a real application of a 3D radiative transfer code based on Monte-Carlo method, [70]. MCFOST code is used extensively to produce synthetic images of the scattered light from disks around young stars. With the unprecedented wealth of data, from optical to radio, fine studies of the dust content and evolution of disks become possible and powerful radiative transfer (RT) codes were needed to fully exploit these data. MCFOST was designed for this reason It was originally developed to model the scattered light in dense dusty media (including linear and circular polarisations), but it was later extended to include dust heating and continuum thermal re-emission. It is currently applied on objects with opacities of high orders of magnitude. This is a rather complex and delicate computation where the codes have to be compared on real conditions.

  The application was specifically constructed to be run upon a lightweight-grid platform configuration. This platform is designed to aid the experimentation of a mesocenter called CIMENT which consists of a number of clusters owned by different laboratories covering various areas of sciences, like biology, medicine, astrophysics, climate prediction, chemistry, etc. As it will be described in more detail on chapter 4, the application was configured to run upon this lightweight grid as multiple sequential independent tasks. The code utilized a random generated number, used as a grain to obtain different internal values to cover a broad

spectrum of results. A specific parameter, that internally defined the number of photon packages, can be modified to change the actual execution time of each task. Furthermore, the number of the submitted tasks can be also defined and the bigger number of tasks we can execute the more reliable are the results.

The configuration of these applications until they can be actually used and provide interesting results for a specific experimentation was not an easy task. In particular, the procedure of reproduction in order to obtain steady and reliable results was rather complicated because demanded tuning on both system and applications side. However, the extra configuration efforts that we had to make were worth the time and work because the actual real-scale computations were able to give us valuable results upon both the astro-physical executions and our systems evaluation.

2.3.4 Evaluation Metrics

The evaluation metrics to validate an experimentation run may vary depending on the exact purpose of the study. A specific optimization technique or a change of a system parameter can affect certain evaluation metrics. Our aim is to be able to capture those evaluation metrics and be able to observe their variations throughout the experimental run. It is expected that a certain optimization upon one evaluation metric may happen to the detriment of another metric. Hence, a post-treatment analysis will be needed in order to evaluate the tradeoffs, gain and losses due to a specifc modification. The final decisions upon the value of a modification will be determined by the analysis of those metrics and tradeoffs.

In this section, we depict the evaluation metrics that were mostly used in our experimentation methodology and we separate them into three different groups according to their characteristics. The first group contains the metrics related to the workload jobs, the second contains metrics related to the system and the last one is composed by the interesting Trade-OFFs that we are willing to explore.

Jobs and applications related Evaluation Metrics

The following metrics are related to the job or the application itself and considering all the jobs of the workload, we can extract either average or total values.

- **Turnaround Time** (or response time), implies the whole duration of a job execution from the submission time until the job is terminated

- **Execution Time**, implies the actual execution of the job from the start until the end of the actual computation. This value also depicts the application performance.
- **Waiting Time**, is the systems response time and implies the amount of time the job will be waiting in the queue until the demanded resources are allocated to it.
- **Slowdown** (or stretch time), consists of the turnaround time normalized by the job's actual running time.
- **Number of Jobs in a certain state**, this value is used for the experimentations that we need to impose a certain duration but we need to acquire the total number of Terminated, Error and still Pending Jobs in order to perform comparisons

System and machines related Evaluation Metrics

The following are system related or machines specific metrics and they can be used either as instant or overall values during the whole duration of the workload.

- **CPU-time Utilization**, is the occupation of specific number of resources for a certain amount of time for all the under execution jobs on the system. This value is measured calculating the number of utilized CPUs with the duration of the whole computation for the specific amount of time.
- **Energy Consumption**, is calculated per machine or per system for a certain amount of time.

2.3.5 Xionee: Workload Trace Analysis, Visualization and Automatic Injection Toolkit

One of the most important parts of the evaluation procedure was related with the analysis and treatment of workload traces. Initially, the need was raised to be able to inject a specific workload trace synthetic or real one, on the cluster for execution. Furthermore, a kind of trace file manipulation was needed in order to obtain a specific part of it with certain characteristics and the smallest possible duration for replay and treatment. The analysis of the workload trace was initially made manually but the complexity of this process suggested us to provide a visualization approach to facilitate this work. Finally, in the end of the workload execution we need to create a new workload log that reflects the actual execution of the injected workload upon the system. Hence, a specific

mechanism is needed for jobs data characteristics extraction and construction of a new workload trace.

Therefore, to answer all those needs we developped a collection of tools, called Xionee [71], to facilitate the evaluation procedure with workload traces. Xionee toolkit is used for the following actions:

- Workload **analysis**, according to a specific evaluation metric for example system utilization or jobs waiting times. This step is effectuated in order to ease the observation of interesting behaviours inside the workload. This can aid either in the initial phase for selection of part of the workload for injection to the system or in the final phase of post-treatment where we observe the actual behaviour of the system.

- Workload **visualization** of specific characteristics for better comprehension of the workload. The visualization extensions have been implemented in order to automate the result figures of the evaluations.

- Workload **treatment**, is the phase of the trace file manipulation in order to extract the particular part of the initial workload log file that is going to be used for injection to the system. This phase is only used for workloads obtained of real production systems.

- Trace **conversion** into a different format. Xionee toolkit currently supports only 3 known formats the swf, the cirne and the one used for esp. This phase is used to convert from one format to another.

- Workload automatic **injection** upon the system, which implies the parsing of the trace file and the construction of a suite of jobs to be submitted on the relevant system. Xionee toolkit supports only 3 different RJMS (OAR, Torque and SLURM) for the moment.

- **Extraction** of jobs characteristics from the database and **construction** of a new workload trace which will reflect the actual execution of the initial workload under the specific conditions provided to the system.

Figure 2.5 shows the procedure steps that we follow for the evaluation

The actual implementation of Xionee was done in Ruby programming language to make advantage of its procedural, entirely object-oriented and functional programming methods. For the analysis and visualization tools we are based on R and gnuplot for the statistical computing part

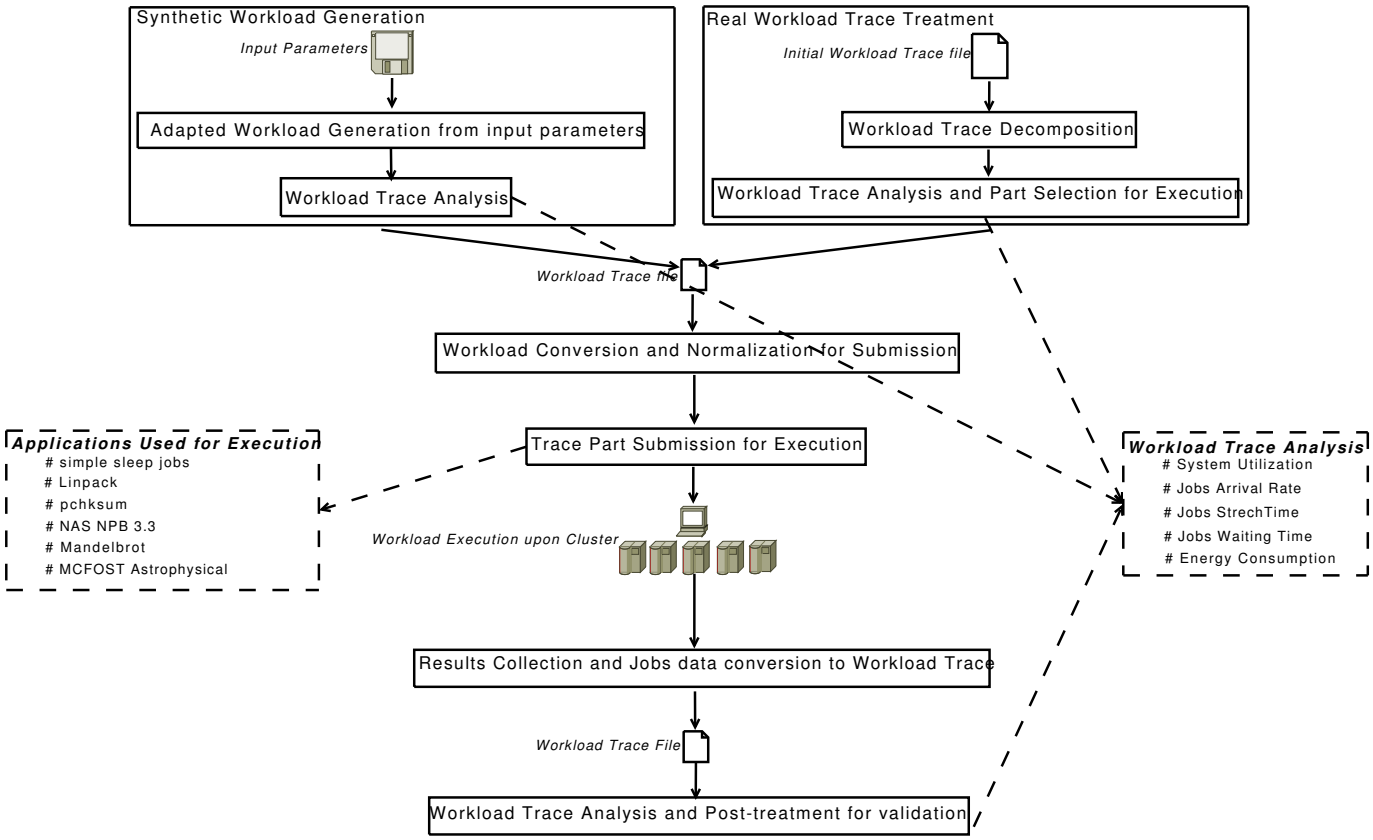


Figure 2.5: Sequence of Xionee evaluation procedure based upon Replay of Workloads.

and graphical representation. Finally, the database extraction tools function with only MySQL DataBases for the moment but PostgreSQL support is also planned for the near future.

The development of Xionee toolkit is deeply related to the system and the evaluation metrics. Indeed, possible changes on the database structure of a supported RJMS, demand a reimplementation of the data extraction toolkit. In the same time the support of a certain evaluation metric needs to be developped upon the analysis and visualization tools. In more detail, the specific analysis and workload trace exploration tool is based upon an histogram implementation taking into consideration specific durations of time and calculating the relevant CPU-time utilization. Following this method, it is easier to select specific parts of the complete workload trace according to their overall utilization percentage and also taking into consideration other metrics like average waiting and execution times.

When a specific instant of start and end of a workload part is demanded then the CPU-time

utilization for this specific duration is caclulated taking in consideration the jobs that have been started before the start of the demanded time interval. Furthermore, this process is taken into account in the injection procedure of the workload trace part upon the system where the automatic submission of jobs is started one hour in advance in order to take care of those jobs that need to be already started for the machines occupation when the actual workload is injected.

2.4 Conclusions

Research upon High Performance Computing uses various methodologies for performance evaluation and experimentation. Middleware like Resource and Job Management Systems are characterized by multifacet, inter-dependent procedures. Therefore an experimental methodology that can measure the performance of the whole system without subtracting particular components, would better reflect reality. This chapter presented the principles of the methodology that we have been using for real-scale experimentation. The main concepts of our approach are the use of a controlled platform for real-scale reproducible experiments along with the utilization of synthetic or real workloads so as to provide the necessary realistic conditions to observe the function of the whole system. In the case of synthetic workloads a particular benchmark called ESP [2] has been adopted in our methodology and specific modifications were proposed to adapt it for evaluation of particular resource management features like topology aware placement. Other important parameters that have to be set are the types of executed applications and the evaluation metrics that are being used. The carefull definition of all these parameters may provide more valuable evaluation results closer to reality. In order to facilitate the analysis, submission, post-treatment and visualization of workload traces for a platform we have developed a particular suite of tools, Xionee, that is used during various phases in our methodology.

The real-scale experimentation can reveile important insights concerning the middleware. Nevertheless we argue that the value of simulated experiments is indisputable and in order to have a complete method for HPC middleware, simulation experiments should be also included. This will enable to validate initial thoughts and intuitive hypothesis and then proceed to the actual implementation and the real-scale experimentation. One of the drawbacks of the presented real-scale methodology is that it is largely influenced by the consistency and reliability of the platform. It does not consider that a failure can occur in the system and if this happens the experiment needs to be reproduced from the beginning. This can result in using a big number of resources, just to provide repetitions of an experiment, that could even lead to incorrect results because of a single

node failure. This can also result to wasted amounts of energy consumption as we will see on chapter 4.

A way to deal with the above reliability and energy consumption issues of the real scale experimentation is to provide the same real methodology upon a virtualized environment [72]. Instead of performing a real-scale experimentation the option of using virtualization techniques can limit significantly the number of real used machines and in the same time even larger scale experimentations can be attained. Finally, the applications used could be enhanced by synthetic ones that have been constructed based on specific profiles, to reflect particular needs of a system. The experiments provided on the following chapters are based upon the above methodology.

Chapter 3

Comparisons of Resource and Job Management Systems

The work of a Resource and Job Management System is to distribute computing power to user jobs within a parallel computing infrastructure. It's goal is to satisfy users demands for computation and achieve a good performance in overall system's utilization 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 take 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 control of execution. In this sense, the work of a RJMS can be decomposed into three main subsystems: Job Management, Scheduling and Resource Management.

In earlier years when parallel computing infrastructures consisted by simple homogeneous architectures (like 1-level flat networks and mono-processor nodes) the function of a Resource and Job Management System was rather straightforward. Its main intelligence and complexity was centered around the efficient scheduling of jobs. However, 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 managers. Furthermore, 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. These issues had to be treated on the job management layer. Hence, both Resource and Job Management layers needed to be reinforced and as a consequence the Scheduling layer, which had to consider more parameters to support those reinforcements, became even more complex.

In the same time the continuous growth of cluster's sizes and computing power still follows Moores law and we have recently observed the entrance on the petaflop scale for supercomputing clusters [3]. The efficient assignment of large number of resources to an evenly large number of users produces issues like job launcher's and scheduler's scalability. Propagation and scheduling algorithms need to be sufficiently equipped to deal with these additional complexities. Moreover, the increase of network diameters and the network contention issues that can be observed in such network sharing scenarios demand a certain treatment so as to favorize the placement of parallel jobs upon groups of nodes which could provide optimal communication speeds. Inevitably the increase of computing resources results in increase of system failures and this is another important issue that has to be treated, in order to guarantee systems responsiveness and robustness. Finally, the great impact upon energy consumption motivated research upon both hardware and software side. Since the RJMS has a constant knowledge of both workloads and resources it can provide techniques for dealing with energy economies taking into account current and future resources availabilities or certain units idleness's.

Hence, the problematics upon RJMS systems have become more complex and in overall the area of resource/job management and scheduling has become an interesting research subject with various extensions.

This chapter provides the state of the art of a resource and job management system (RJMS). It presents its global architecture design and discusses challenges and issues that have emerged through the latest technological evolutions, the applications needs in computing power and the continuous growth of cluster sizes. It provides a survey of some of the most known commercial and opensource RJMS and presents a non-exhaustive analysis of their principal concepts and their approaches to deal with the latest evolutions and needs.

3.1 Background and Related Work

Since the beginning of the first RJMS back in the 80s, a lot of software packages have been proposed in the area to serve these needs. Most older systems like NQS, DQS, Utopia has been initiated as university research projects that later became industrial software. Figure 3.1 provides a chronological map of most of the known RJMS since the beginning of the research in the domain. For example PBSPro which is a commercial solution has started as a university effort in the USA with the name OpenPBS. The same software then turned into commercial with PBSPro and an open-source solution based on OpenPBS was maintained under the name Torque. Another widely

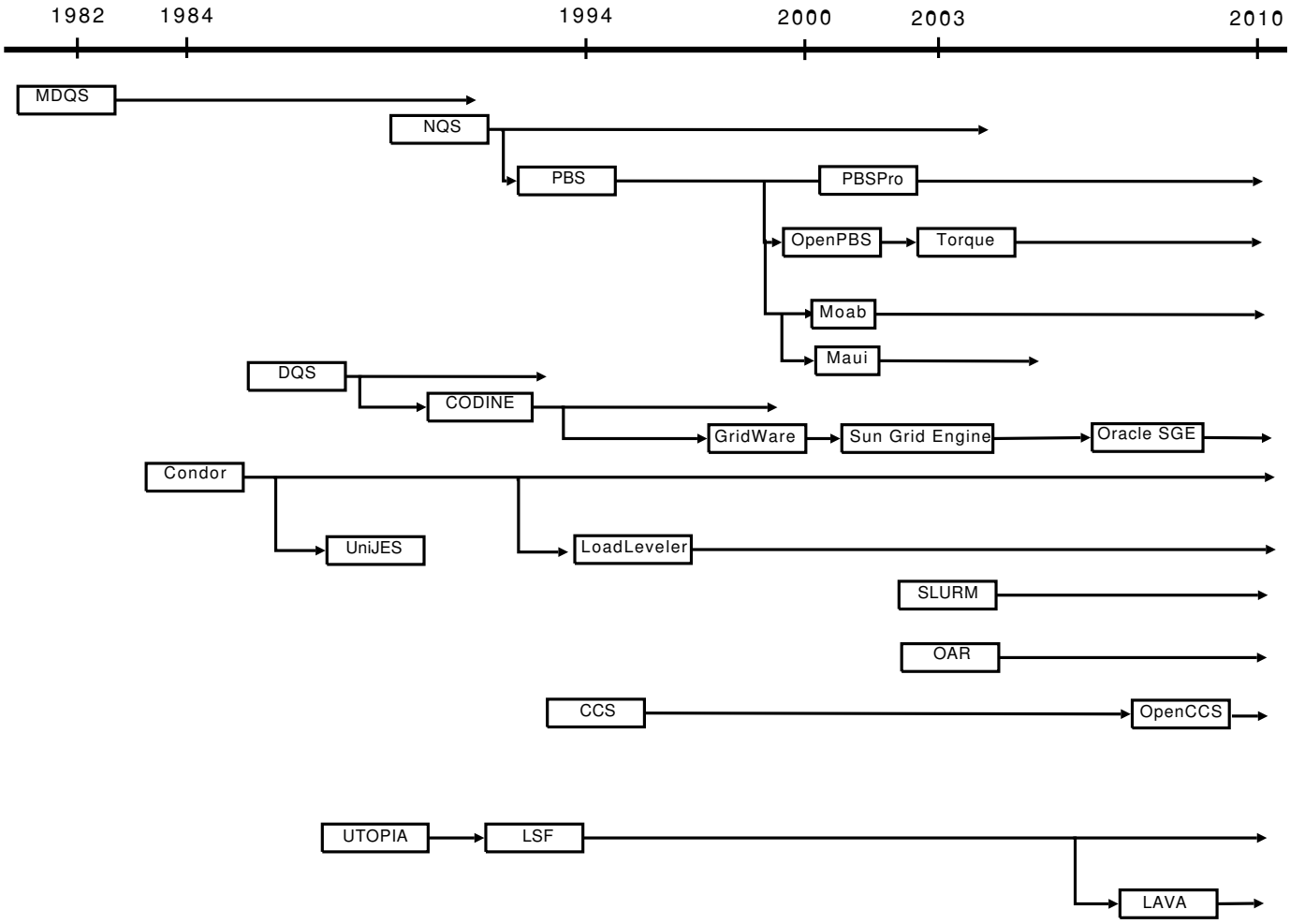


Figure 3.1: Resource and Job Management Systems Chronological Map

known open-source RJMS system is Condor which is also one of the oldest in the area and remains an innovative research tool and a powerfull production system. It is interesting to see that there was a wave of new generation systems like SLURM, SGE and OAR which started after 2000. Section 4 provides extensive information and descriptions about some of the above RJMS.

In addition a lot of studies have been made to evaluate and compare those different approaches [73], [74], [75]. One of the first studies which regrouped and evaluated various RJMS was published on June 1994. In this article [73] Kaplan et al. set some evaluation criteria and analyzed the concepts and main features of some of the most common RJMS of the time. In [75] Jones and Brickell made a measured conceptual comparison of specific evaluation criteria, according to their

needs for a RJMS system for a production cluster on NASA laboratories. Yonghong Yan et al. in [76] have made a comparative study between the most known commercial and one opensource RJMS. This study analyzed architecture design and principal functionalities of SGE, LSF, PBS Pro and LoadLeveler.

One of the first studies of performance evaluations for RJMS appeared as a technical report [77] and later as an international publication [78]. 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 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. The utilization was measured by calculating the average percentage of the CPU time used by all the jobs upon each computing node of the system. These measures were then averaged over all computing nodes. This was a method for understanding the function of the scheduler and evaluating its efficiency.

Another empirical study [79] measured throughput with submission of large number of jobs and efficiency of the launcher and scheduler by using a specific ESP benchmark [56] also described on chapter 2. 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 [78]. 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. Finally, the work in [2] provides functionalities comparison between some opensource RJMS along with performance evaluation of scheduling and launching efficiency of these systems, through the use of ESP benchmark.

3.2 RJMS Principles

In our study we divided the principles of a Resource and Job Management System into 3 categories. The first category is constituted by principles related to resources and their treatment, like resources selection, matching of resources with jobs, job propagation and finally binding of tasks upon resources. The Job Management subsystem is defined by principles related to jobs description, control, monitoring and quality of services. The last category is composed by principles like queues management and scheduling algorithms.

Table 3.2 presents the different abstraction layers of a Resource and Job Management System.

RJMS subsystems	Principal Concepts	Advanced Features
Resource Management	-Resource Treatment (hierarchy, partitions,..) -Job Launching, Propagation, Execution control -Task Placement (topology,binding,..)	- High Availability - Energy Efficiency - Topology aware placement
Job Management	-Job declaration (types, characteristics,...) -Job Control (signaling, reprioritizing,...) -Monitoring (reporting, visualization,..)	- Authentication (limitations, security,..) - QOS (checkpoint, suspend, accounting,...) - Interfacing (MPI libraries, debuggers, APIs,..)
Scheduling	-Scheduling Algorithms (builtin, external,..) -Queues Management (priorities,multiple,..)	- Advanced Reservation - Application Licenses

Table 3.1: General Principles of a Resource and Job Management System

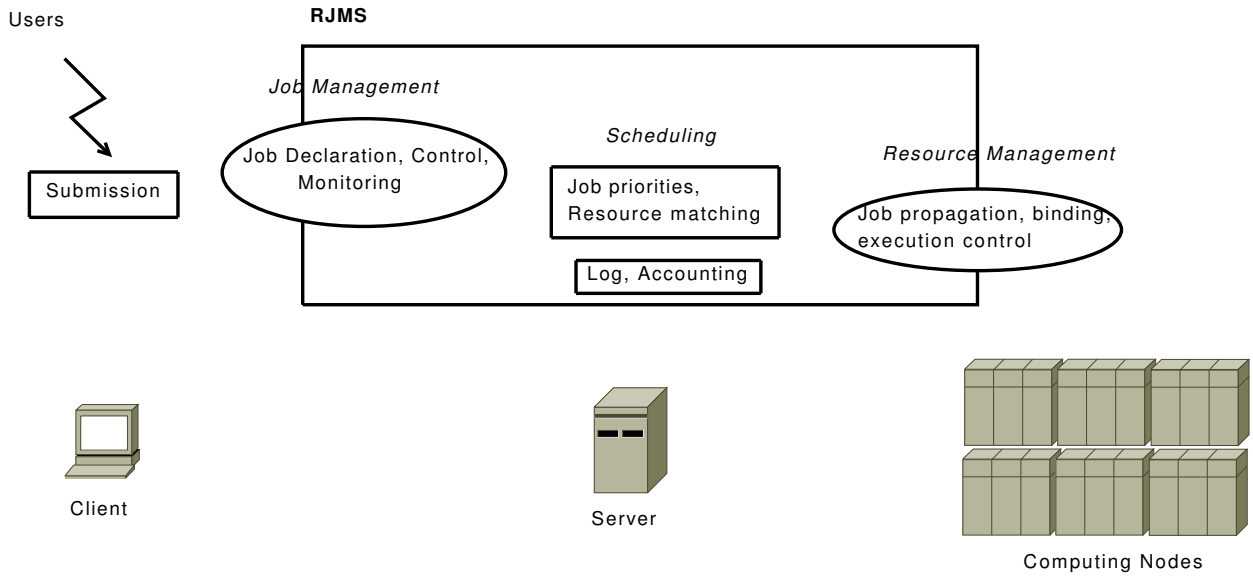


Figure 3.2: Global architecture of a Resource and Job Management System

Figure 3.2 presents the global architecture of a Resource and Job Management System. It shows the distribution of the different tasks presented on table 3.2 along the different components of the infrastructure. The scheduling tasks are specifically executed on the central part of the RJMS

system which is the server. The Resource Management tasks demand a cooperation between the Server side and the Computing Nodes. This is usually managed by specific RJMS daemons that exist on the computing nodes and provide a constant communication and information exchange between the Server and the Computing Node. Finally the Job Management tasks are launched and controlled by the client user side but the main part of the work remains on the server side. Figure 3.3 shows the sequence diagram of a job from the submission until the end of its execution.

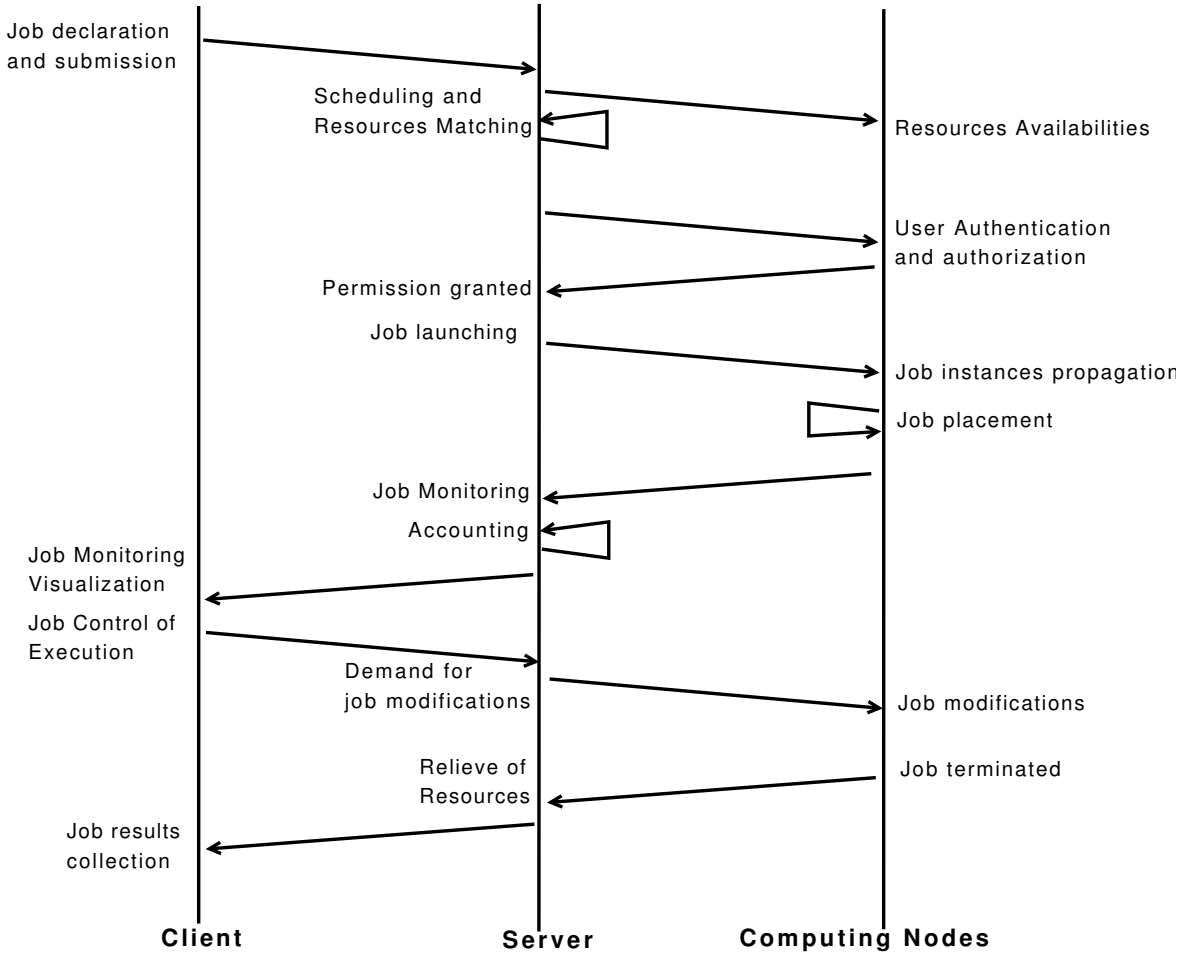


Figure 3.3: Job sequence diagram

The following subsections provide some details concerning the above principal tasks of a Resource and Job Management System.

3.2.1 Resource Management

The resource management (RM) abstraction layer of the RJMS is responsible to collect and provide all information concerning the computational elements (resources) of the cluster. This information has to be delivered to the scheduler to initiate the job scheduling and to the user or administrator to inform about the availability and the state of the cluster. A fast and error-less delivery of information will guarantee the efficiency and robustness of the system.

In more detail the Resource Management subsystem is responsible for the following basic tasks:

Resource Treatment

This task is defined by the characterization of the cluster resources, according to their components. In earlier years clusters were mostly homogeneous infrastructures composed by similar resources. However, in latest platforms the degree of heterogeneity has been increased and clusters are more often composed by groups of resources with different hardware components. Widely used techniques to deal with cluster heterogeneity are the definition of different *partitions* in the same cluster or the support of *multi-cluster* for one single RJMS. Furthermore, the current evolutions in processors (multicore, GPU) and communication networks, have introduced new levels of hierarchies inside the node (node/cpu/core/thread) and inside the network (different topologies and levels of hierarchy of switches and nodes interconnections) that need to be taken into account by the RJMS for a better cluster exploitation.

Some Resource Manager systems propose *hierarchical view* of resources. This provides finer resources treatment and clearer view of the cluster topology to the user. On the other hand, the hierearchical treatment of resources definitely increase the complexity of the algorithms related to command propagation and scheduling. Advanced RJMS provide techniques that can disable the hierarchical view of resources using only the node as smaller computational unit, giving the opportunity to large systems for performance optimizations.

Task launching and execution control

The Resource Manager subsystem waits the specific information concerning the job characteristics, from the scheduler and the job management layers. The procedure can then be decomposed in two steps. At first, it initiates the launching on the distant resources by utilizing a specific *propagation technique*. This technique, which varies according to the Resource Manager's design, is either based on distant execution commands (rsh,ssh) or on intermidiate software (daemons on each computing node) or even on a kind of combination of both. This procedure for task and command propagation upon specific resources is also used for execution control, resources sanity check and

generally all the tasks that implie a command deployment upon a specific number of resources. The use of specific optimized techniques like tree broadcast can ameliorate the propagation time and most of the advanced Resource Manager systems make use of this kind of optimized techniques, which are analyzed in more detail on section 2.3. The second step concerns the actual creation of the task upon the distant resources. This includes the environment variables propagation along with the different parameters of execution depending on the options selected by the user.

RJMS systems support both simple and parallel tasks on which the procedure for launching differs. The launching of parallel tasks follow a more complicated procedure than the simple tasks. The procedure contains one more step which is the synchronization of tasks, in particular the synchronization of the communication software (MPI libraries). During this process the execution of the initialization function MPI_Init() demands the simultaneous start of all tasks on all the nodes. This process can become even more complicated if we add aspects like secure authentication or input data transfer before the start of the jobs. Normally, in modern platforms where two or more communication networks (Ethernet, Infiniband) exist, the process of task launching makes use of Ethernet for the communication establishment (master-computing nodes) and the high-speed network for the inter-nodes (MPI) communication.

Task placement

In the earlier years where computing nodes were single cpu machines and only one job could allocate a single node there was no need for process confinement upon the node. The emergence of multicore architectures that introduced new levels of hierarchies inside the node arose the need for methods of *job confinement* upon specific allocated resources in a single node. Indeed, multiple users can find themselves working on the same computing node, which raised the need of task placement upon specific cores in order to avoid collisions between different tasks. The problem of collisions are that cores or socket resources easily can be oversubscribed, resulting in degraded performance, while other sockets or cores of the same node stay idle. Hence, the secure placement upon cores and in general specific parts of the resources (memory, disk I/O, bandwidth) became an important challenge for the new RM systems. OS techniques like CPUSETs, sched_affinity are supported by the Resource Managers to provide a fine management of the CPUs (cores) of the nodes. Concerning the memory a used confinement technique implies the declaration of usage limits depending the number of used cores.

Resource Management Advanced Features

The important features of the resource management layer are directly related to latest technological evolutions and the new needs that have arisen because of them. Fault-tolerance mechanisms

are needed to guarantee the *high-availability* of the cluster in case of hardware failures. *Topology aware scheduling* favors the choice of resources which may provide performance optimizations either because of network topology characteristics or due to internal node hierarchies (multicore, NUMA). Finally, *energy efficient resource management* techniques needed to be developed for energy economies on large systems.

3.2.2 Job Management

The job management subsystem provides the means for definition, submission and monitoring of jobs upon the RJMS. It is responsible for the tasks related to the declaration and control of users jobs. It possesses features to ease the use of the cluster and should provide transparency of the complex internal functions of the RJMS. It may support different types of jobs (batch, interactive, array, ...) with various characteristics and ways to declare the needed resources in varied complexity according to the users needs. Tasks like suspend/resume, checkpoint/restart and file stage in and out are initiated and controlled by the Job Management subsystem. Figure 3.4 shows the different states that a job gets from its submission until the end of execution.

Job declaration

All RJMS provide ways (commands or web interfaces) for the users to describe their jobs characteristics and select the resources of their preference. There are different types of jobs with different functionalities. For example, *interactive* jobs are used when the user needs to be connected directly upon a node (shell) and launch its experiment manually. On the other hand, *batch* jobs are used for direct script execution upon the allocated computing nodes. A specific kind of multiparametric jobs called *array* or *bulk* jobs provide a way for using a big number of small jobs (in terms of demanded resources), if a large job can be decomposed into smaller ones. This may result in faster scheduling and better overall turnaround times. Apart the typical type of jobs it seams that special type of jobs can be also met into various RJMS. The *besteffort* type provides a lowest priority job that is killed whenever a normal job demands the resource and it is used into global computing contexts.

Job Control

Once the job is submitted to the cluster then the scheduling and the resources binding procedure take place. Some RJMS provide functionalities that allow the user to modify some initial options (like walltime, standard output files, reprioritizing) while the jobs are submitted but still waiting for execution. The possibility to interfere with jobs during their execution is also provided by certain RJMS. Specific commands that can send signals on jobs to perform specific programmed actions

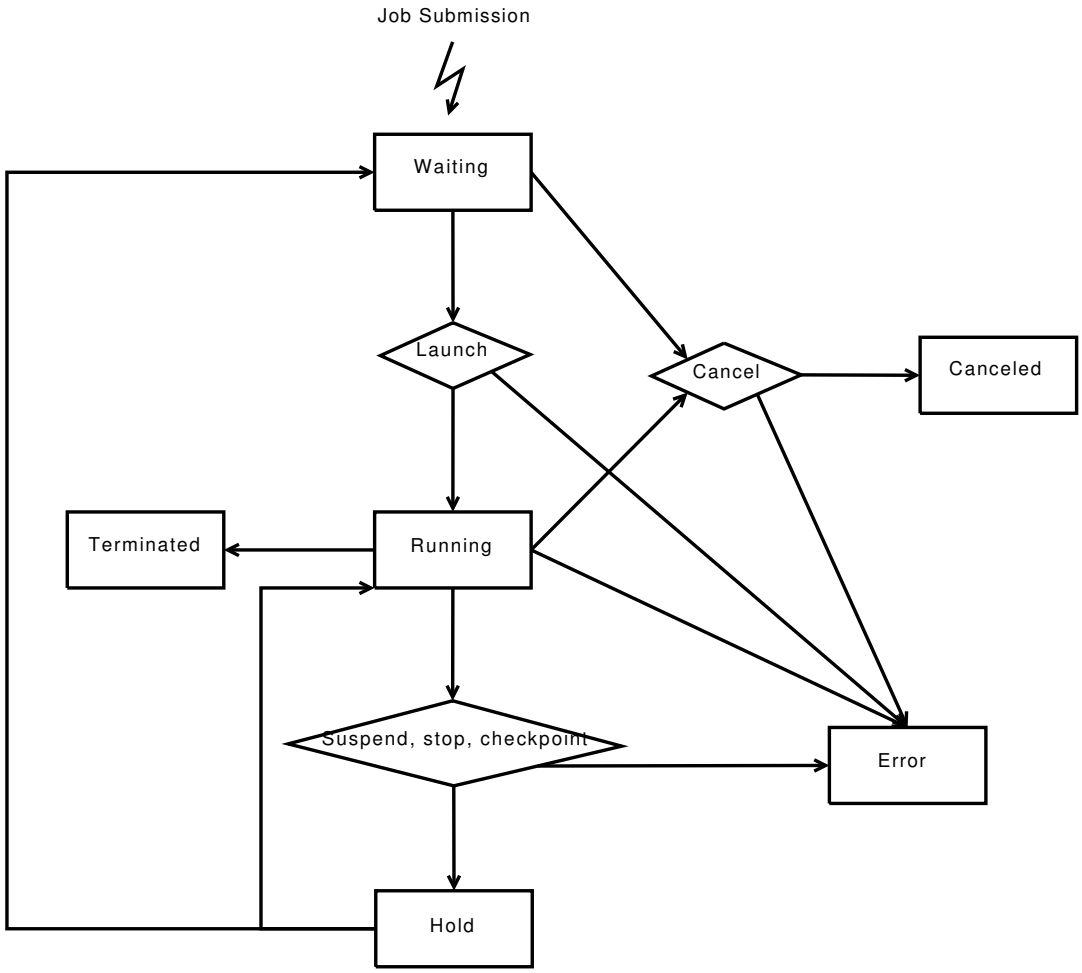


Figure 3.4: State diagram of jobs in a RJMS

(like resources folding or expanding in case of disponibilities).

A submitted job may need specific treatment just before the start of its execution or immediately after the end of it just before the resources are relieved from the job. The user may make use of specific *prolog* and *epilog* scripts that can be used to perform programmed actions like shell customization or environmental variables charging. Furthermore, in the same context jobs may need to load data to start their execution. The functionality called *stage in* and *stage out* can be used

Job Monitoring and Visualization

Specific job management tasks exist to follow the execution of jobs upon the cluster. Furthermore, explicit *visualization* interfaces may be used to monitor the execution of the jobs along with

the state of different cluster resources.

All information concerning the jobs execution upon the cluster (like duration, allocated resources, energy consumed, data transfered, etc) should be logged and stored (usually on special databases). This saved data may be used for specific statistical analysis and *reporting* to the administrators and users.

Job Management Advanced Features

Some of the most advanced Job Management features. are related with *quality of services* (QOS) tasks. Once a job is submitted and started its execution, the user or administrator may have the possibility of suspending it on RAM and resuming it later. This functionality, called *preemption*, can be either asked specifically on the submission command or provided as a scheduling algorithm as described on section 2.4.

The functionality of *checkpoint/restart* is a widely used method for fault-tolerance upon HPC clusters. In case of failure the application will be able to restart itself from a stored checkpoint and gain valuable time of computation. Both application and system level checkpoint/restart implementations may be supported.

Authentication and *authorization* services are used for increased security mechanisms. Software like Munge or Kerberos may be used to guarantee secure communication between nodes.

The RJMS can be interconnected with various exterior systems or libraries. All kind of *interfacing*, with parallel programming environment libraries (like OpenMPI), parallel debuggers (like STCI) application programming interface (like DRMAA), or even monitoring services (like NWS) may take place.

3.2.3 Scheduling

The part of the RJMS which hides all the intelligence of the system is the scheduler. Its main role is to assign jobs according to the users needs and predefined rules and policies, upon available computational resources that match with the demands. A typical scheduler functions in cooperation with queues which are elements defined to organize jobs according to similar characteristics (for example priorities).

Scheduling Algorithms

Motivated by needs for optimized cluster utilization or fairness of the cluster resources among its users; different kind of scheduling policies have been defined and implemented upon RJMS systems. Table 3.2.3 provides the definitions of existing scheduling policies and figures 3.5 present examples of function of each algorithm. The most typical policy used in this context is the FIFO (or

FCFS). A rather classical optimization as described on 3.2.3 is the *backfilling*, which has several versions like *conservative* or `aggressive`. The first one which is most commonly used a smaller job is moved forward in the queue as long as it does not delay any previously queued job. The second also known as (*EASY*) backfilling a small job is allowed to leap forward as long as it does not delay the first job in the queue. *Gang Scheduling* policy is a stricter variant of *Coscheduling* and *TimeSharing* allows the actual concurrent execution of jobs upon the same resources. The *fairshare* policy tries to provide a fair utilization of resources and finally *preemption* provides the ability to prioritize the workload on a cluster.

Scheduling Policy	**Description**
FIFO	jobs are treated with the order they arrive.
Backfill	fill up empty wholes in the scheduling tables without modifying the order or the execution of previous submitted jobs.
Gang Scheduling	multiple jobs may be allocated to the same resources and are alternately suspended/resumed letting only one of them at a time have dedicated use of those resources, for a predefined duration.
TimeSharing	multiple jobs may be allocated to the same resources allowing the sharing of computational resources. The sharing is managed by the scheduler of the operating system
Fairshare	take into account past executed jobs of each user and give priorities to users that have been less using the cluster.
Preemption	suspending one or more ”low-priority” jobs to let a ”high-priority” job run uninterrupted until it completes

Table 3.2: Common Scheduling policies for RJMS

Queues Management

The RJMS may provide multiple queues (or partitions) which can receive jobs for executions with similar characteristics. Usually the RJMS has one single scheduler. Advanced scheduling

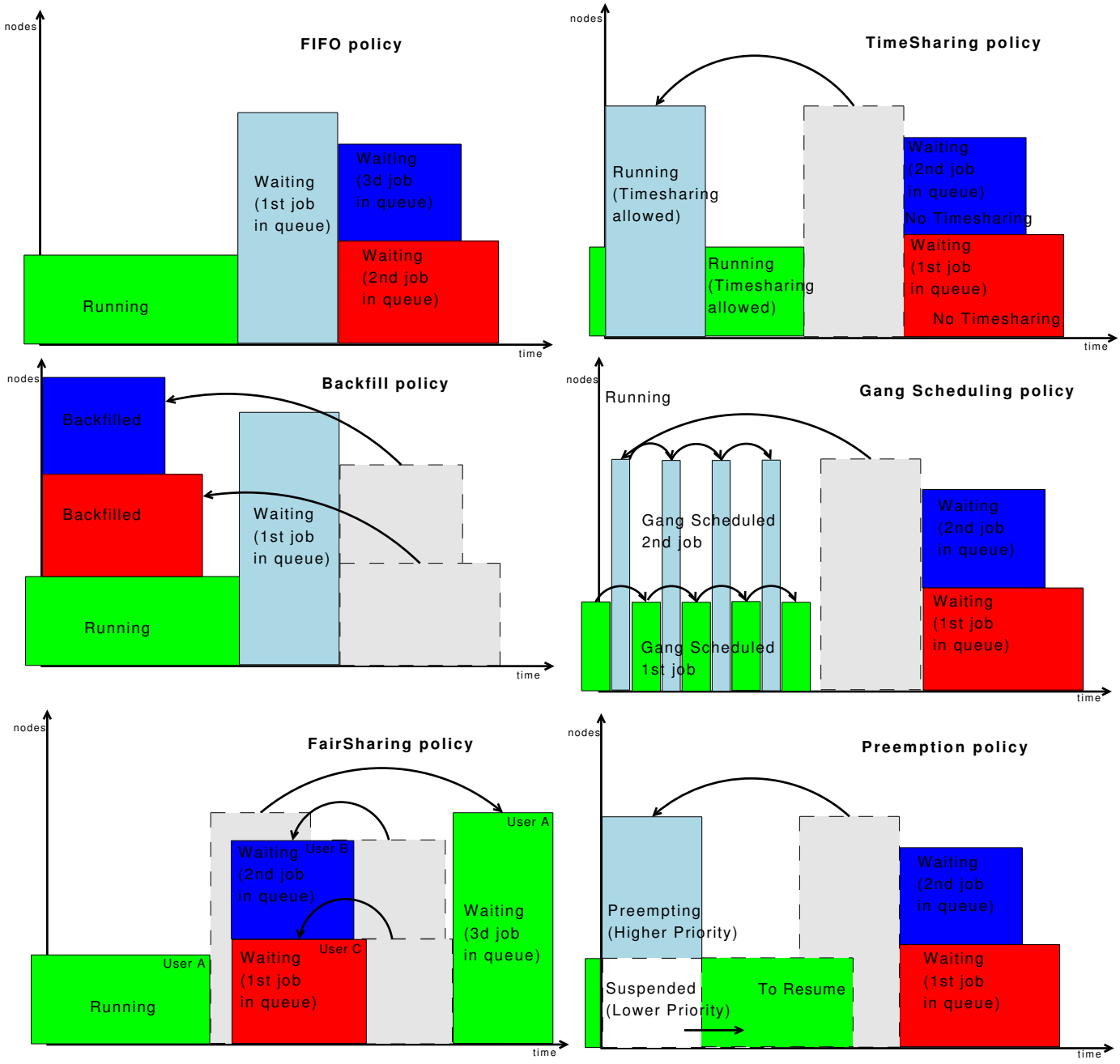


Figure 3.5: Function examples of Scheduling policies

subsystems may allow the use of a different scheduling policy providing a better quality of services to the users.

Scheduling Subsystem Advanced Features

Specific types of applications (i.e. ISV) need a kind of license so as to start their execution. If there are few licenses for the use of a specific application upon a cluster then tokens will be distributed and each user will have the opportunity to execute the application on each turn. The explicit *management of licenses* tokens is supervised and performed by the scheduler.

A very usefull functionality implemented on various RJMS is the *advanced reservation* of resources. It is an indispensable feature for ease of use upon the cluster but should be used with moderation because it may become a bottleneck under specific conditions. Users can plan in advance their computations without having to worry when their job is going to start. Nevertheless reservations is an known enemy of scheduling. Indeed, big number of reservations may result in underutilization of the cluster, because the optimized algorithms of the scheduler cannot function.

3.3 Research challenges

The recent appearance of the first petascale supercomputers and the new roadmaps for exascale platforms before the end of the decade have raised the need for research upon specific issues related to the scalability of Resource and Job Management Systems. Issues like *topology aware placement*, *launcher and scheduler efficiency*, *system high availability* and *energy efficiency* are some of the subjects that receive most of the attention, by the scientists, in the area.

3.3.1 Topology aware Placement

The size scaling of those infrastructures take place by increasing the number of nodes along with the number of cores per nodes (multicore processors). The first increase results into bigger network diameters and the second into bigger depths inside each single node. Both facts dictate the need for finer knowledge of the topology of the platform in order to have better treatment of resources and efficient system exploitation.

Internal Node Topology

The continuous scaling of multicore processors with shared caches and non uniform memory access causes the internal node topology to become increasingly complex. The increasing complexity and level of parallelism inside the computing nodes raises the question of how to schedule work

so as to minimize the impact of this complexity. Applications using OpenMP, MPI or hybrid models (OpenMP+MPI) have to be carefully placed upon the machines so that hardware affinities are efficiently handled for optimal performance. Shared-memory or synchronization between tasks benefits from shared caches, while intensive memory access benefits from local memory allocations. Exploiting modern architectures thus requires an in-depth knowledge of the internal node topology, but also of the relevant application behavior. The RJMS can play a significant role in this situation since it maintains information from both applications and resources.

An important problem that may arise in case of non sufficient treatment of the hardware topology is *internal fragmentation* which is the phenomenon that results into idle processors when more CPUs are allocated to a job, that it requests. Another issue is the efficient task placement. Some parallel applications benefit when their processes are distributed to different sockets on the same node, while others, when they are placed upon a single socket and different cores. Thus, ideally the RJMS should provide automatic optimal task placement techniques based on the hardware affinities along with ways to allow the choice of specific placement that would provide the best performance for his application [80].

In latest version of Linux kernels it is possible to specifically tell to the scheduler which process can run upon which core. This core affinity (or core binding) as it is called, can be influenced, in Linux, via 2 different ways: Either a system call which takes a bitmask as parameter (sched_affinity [81]), where each bit reflects one core and if a core is on with bit value 1 (or off with value 0) then the scheduler does (or not) migrate the process to that core; Or using processor sets (cpusets [82]) which are defined by a hierarchical pseudo-filesystem upon which only processes bound to this set can be executed. The mask or set is inherited by all child threads/processes that means that all of them will be using the same set of cores.

The use of any of this techniques allows the RJMS to have full control of the processes of its jobs and permits the dedicated binding of jobs upon specific cores along with the efficient cleaning of remaining processes after the end of the job. An extension of cpuset which has recently been added to the Linux kernel (after 2.6.32) are the control groups [83]. Each control group is a set of tasks on a system that have been grouped together to better manage their interaction with system hardware. They provide a mechanism for aggregating/partitioning sets of tasks and all their future children, into hierarchical groups with specialized behaviour. This behaviour is defined by the different subsystems that exist like: *cpuset* which assigns the tasks upon cpus, *memory* which sets limits on memory use, *freezer* which suspends or resumes tasks, *devices* which allows or denies access for tasks to specific devices and *blkio* (supported after kernel version 2.6.35) which assigns

specific amount of IO or even network bandwidth to tasks. The intention is that different kind of subsystems hook into the generic cgroup support to provide new attributes for cgroups and therefore to the user's tasks.

Closer to the application level, and in order to simplify the access to this hierarchical hardware topology, which is different with kernel versions and Linux distributions, external APIs are used as intermidiate layer for core affinity settings. Portable Hardware Locality (hwloc) project [84] is an API which provides an abstraction of the hierarchical topology of modern architectures, including NUMA memory nodes, sockets,shared caches,cores and simultaneous multithreading and can allow applications to control topology affinities according to their internal communication patterns. This project is a merge of two older different APIs libtopology [85] and PLPA [86] (protable linux processor affinity).

Any technique of the above can be adopted by the RJMS. The best approach could be to have a first confinement induced by the RJMS with cpuset or cgroups and at a second time allow the use of an API, like hwloc, for specific applications that could benefit from further topology affinities and finer granularites. In Linux, chlid-processes are allowed to re-define their core affinity.

All these kernel mechanisms used for task confinement do not induce (normally) any overhead on the RJMS system and they can be applied instantly upon a single node. The bottleneck for task binding upon a big number of nodes remains only upon the task propagation algorithm. Chapter 4 gives a thorough analysis of the strength of the cpuset mechanism.

Network Topology

The increase of the cluster in terms of computing nodes has as result the use of large network diameters to connect all of those nodes. Computation has become cheap but communication on the network is becoming the bottleneck for scaling of parallel applications. Network contention, specifically, is becoming an increasingly important factor affecting overall performance [87]. Highly parallel applications are sensitive to communication performance . The distance between nodes may play an important role in case of parallel computation. Depending on the communication pattern of the application, and the way processes are mapped onto the network, severe delays may appear due to network contention; delays that result in longer execution times. Nodes that are connected upon the same switch will result in better parallel computing performance than nodes that are connected on different switches. Mapping of tasks in a parallel application to the physical processors on a machine, based on the communication topology can lead to performance improvements [88]. Different solutions exist to deal with those issues on the resource managemement

level.

We are especially interested for fat-tree network topologies which are structures with processing nodes at the leaves and switches at the intermidiate nodes [89]. As we go up the tree from the leaves, the available bandwidth on the links increases, making the links ”fatter”. For the sake of simplicity a group of nodes connected upon the same switch is called an *island* of nodes for the rest of the thesis. Network topology characteristics can be taken into account by the scheduler [90], [91], [92] so as to favor the choice of group of nodes that are placed on the same network level, connected under the same network switch or even placed close to each other so as to avoid long distance communications. This feature becomes indispensable in the case of platforms which are constructed upon pruned butterfly networks [93], [94] where no direct communication exist between all the nodes. This reduces the number of fast communication group of nodes to only those connected under the same switches. Hence, the efficient network placement could be treated as an extra scheduling parameter. This will definitely provide an overhead upon scheduling, resulting in longer waiting times for pending jobs on the queue, but will eventually consist in improving performances of single parallel applications which means that once the scheduling decision has been made the turnaround time will be faster. On the other hand jobs that are not communication bound will have to endure the overhead even if they will not profit of faster turnaround times. So the waiting times for jobs can be expected to increase, due to external fragmentation [95] . External fragmentation [96], [95] is the phenomenon that exists when a sufficient number of processors are available to satisfy a request, but they cannot be allocated contiguously.

Alternatively, another solution to deal with this waiting time problem for non-communication bound applications is to particularly declare the interest for topology aware placement on the job level. Therefore it becomes the choice of the user and hence only sensitive applications will profit of the optimization and endure the overhead on waiting times. However this increases the risk for external fragmentation, since the job will be waiting for the right processors to be liberated.

3.3.2 High Availability

The increase of number of resources along with the application needs for computing power have turned the system reliability into an important factor for Resource and Job Management Systems. This becomes an urgent issue in the case of large clusters where the MTTF (Mean Time To Failure) is getting lower and the MTTR (Mean Time To Repair) is increasing. The RJMS can deal with those hardware failures by providing solutions on two different levels. One is to deal with the central server failures in order to guarantee the system's high availability. The second is to treat

the computing nodes failures especially on worst case where a job is executing upon it.

System High Availability

In case of cluster resources failures the RJMS system should be able to respond and treat the desfunctioning to guarantee a robust service. Some significant research has been made in this area [97], [98], [99], [100]. The most important component that needs to be well protected is the central manager which is usually lying upon a central server node. System's high availability can be ensured by having a configuration of backup server that can take the control when the primary doesn't respond any more. Some systems have even went one step further and have techniques to replace the backup server as well. Computing nodes could take the place of this secondary server in case of further failure. place of this secondary server in case of further failure. Specific event handling techniques are used for monitoring, reporting and even automatic correction of problems. The monitoring of resources with expulsion of suspected for error nodes can assure the health of the whole system.

Job Fault Tolerance

Hours of valuable calculations upon a very big number of nodes can get lost due to a node failure. Apart the numerous efforts by HPC vendors to improve reliability on hardware side continuous research projects are trying to enhance recovering techniques. Resource and Job Management Systems can support different kinds of recovering techniques. The most common are either System or Application level checkpoint/restart approaches. For the system level checkpoint/restart approaches the integration with libraries like BLCR, DMTCP or other is quite common. For the application level recovering techniques their support implie the issue of specific signals periodically towards the application in order to proceed with a checkpoint procedure. In case of failure the job will be automatically restarted and if a checkpoint exists it will start from the latest checkpoint.

More details upon recovering techniques and specifically system level checkpoint/restart approaches can be found on chapter 3.

3.3.3 Energy Consumption

The energy consumption of HPC clusters has become an important parameter that needs special consideration. This need becomes indispensable in large scale infrastructures. The main motivation is the electrical and financial economies for the overall platform consumption. The Resource

and Job Management System can combine information from both hardware components and users workloads to provide techniques that can reduce the overall energy consumption.

The RJMS can collect and transfer to the scheduler the consumed energy information of different hardware components. The analysis of this information with corellation with the users workload may be the input for specific developed algorithms that result into new energy-aware scheduling policies. Those algorithms may take into account the clusters unutilized computing nodes and take relevant action (reducing CPU frequency, hibernating or even shutting down specific nodes). Ofcourse the energy management is directly related to each sites workload, thus the RJMS should provide possibilities for energy management policies customization. Furthermore the collected energy consumed may be used for jobs and users accounting. Chapter 6 provides thorough analysis of the research directions upon the subject of energy efficiency through Resource and Job Management Systems along with experimental results.

3.3.4 Launcher and Scheduler

Finally two parts of the system that are greatly influenced by the increase of nodes and resources are the launcher and scheduler of the RJMS.

Propagation techniques

The increase of number of nodes has a significant impact upon the communication and command propagation between the master and the computing nodes. Most RJMS systems provide job launching through socket communication between the nodes. For the propagation technique, some of them, are based upon a locally parallelized deployment that connects to the client daemons installed on all the remote nodes like mpirun [101]. This schema provides good scalability but its performance varies between MPI implementations. Systems like SLURM [102] resource manager or gexec [103] deployment tool provide optimizations to the above schema by using dynamically adapted deployment based upon tree structures. Following this schema, the master node establishes socket communication with a specific number of nodes which by their turn communicate further to other nodes until all nodes have established connection, forming a tree structure. Similar approach are used upon pdsh [104] remote shell toolkit which uses a ”sliding window” parallel algorithm to conserve socket resources on the initiating node and to allow progress to continue while timeouts occur on some connections. These algorithms have been proven to be scalable to thousands of nodes, however they are not well suited to heterogeneous platforms or multicluster infrastructures.

Cyrill Martin has made a thorough study upon propagation techniques and remote execution deployment [105]. This study has resulted in the conception of an adaptive propagation technique based upon standard Linux protocols like ssh and rsh. The resulted tool named Taktuk, initially implemented on C and later on Perl [106], [107]. This tool for large scale remote executions deployment, provides optimized propagation techniques using a dynamic adaptation mechanism based on work-stealing that balances deployment tasks between local parallelization and remote distribution. This load balancing strategy adapts to load variation in the network as well as in the nodes. Thanks to the dynamic adaptation mechanism and to the resulting efficiency and scalability it is suited to interactive parallel tasks on both homogeneous and heterogeneous machines.

Other tools base their propagation optimizations upon hardware capabilities to solve the issues related to deployment. This is the case of Storm resource management system [108] which relies on Quadrics Network Cards to provide high-speed communications and broadcast capabilities to deploy in constant time. Besides the portability limitations, Storm is highly scalable and efficient and has been prooven to be some orders of magnitude faster than most other deployment tools [109].

Scheduling

Scheduling upon parallel and distributed systems is a complex task that has attained a lot of focus by researchers [110], [111] since the first appearance of those systems. The architectural evolutions with the deep hierarchies and the increase of computing resources along with the user and application needs triggers new challenges and complexity the process of scheduling for a resource and job management system [112].

The scheduler is the central point of intelligence of the RJMS system. The more advanced features are supported upon the RJMS the more complex will be 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. Moreover, this complexity will be increasing with the size scaling of the cluster. The heterogeneity of clusters can complicate further the job of a scheduler since tasks that require a certain set of resources may only be schedulable on a small subset of the nodes in the system. The impact of advanced reservations can be significant in waiting times and overall system utilization but this is greatly influenced by the workloads as well [113].

A lot of research with valuable results has been conducted, that compares scheduling algorithms in various contexts [111], [111].

As already described FCFS (or FIFO) manages jobs with the order they arrive. SJF (Shortest Job First) and LJF (Longest Job First) are the simpler variations which can provide better efficiency under specific job mixes.

A more sophisticated algorithm that has prooven good performance in most of the cases is Backfilling. A lot of variations exist for this algorithm like conservative, aggressive (EASY) [114] or specific parameter dependent like slack based [115]. In general aggressive backfilling results into better system utilization than conservative [116] but the second one is more widely used thanks to it's fairness and guarantee of services. Nissimov and Feitelson developed another kind of backfilling called probabilistic backfilling [117] which is based upon self on-line-learning prediction mechanisms of user job runtimes. The method improves its predictions as more jobs of the workload are submitted and terminated. This policy has shown even better results than aggressive backfilling.

A widely alternative to backfilling policies is proposed by Gang Scheduling. The notion was firstly introduced by Oustrhout and the concept of jobs coscheduling [118]. As a subsequent class of coscheduling, gang scheduling temporarily preempts and then reschedules jobs upon specific time intervals. It provides an environment similar to a dedicated machine, in which all job's processes are executed together and at the same time resources are time-shared among different jobs. In particular, preemption (suspend/resume) is used to improve performance in face of unknown runtimes. Gang-scheduling has been reported to have very good performance efficiency when compared to backfilling [119]. The good performance efficiency of gang-scheduling is also due to the fact that we can overlap the communication phase of some jobs with the computation phase of others [120] and hence enable the execution of pairs of jobs that will result into faster turnaround times and better system efficiency. Nevertheless not all application can perfectly fit into these categories that's why in cases of high memory requirements hybrid techniques have been proposed [120].

The parameter of Preemption [121] can also be used for job scheduling to optimize FIFO or Backfilling techniques [122]. The preemption is defined by the stop and later restart of lower priority jobs in order to allow higher priority jobs perform urgent computations. Preemption can be implemented with stop/reschedule, suspend/resume or checkpoint/restart models.

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, the platform architecture and the workload.

The hierarchical management of resources provides great flexibility for an RJMS upon SMP or NUMA architectures. Unfortunately, this comes in expense of a more complexified scheduling since the number of resources to be considered is much bigger and hence the scheduling process has more alternatives to take into account. A possible solution could be to provide scheduling depending on the resources granularity (switches/nodes/cpus/cores) which means that depending the needs the scheduler could particularly set it on nodes level to speed the scheduling process or to cores level to have finer treatment. Separating resources on smaller partitions and performing scheduling upon those specific partitions may also provide better scalability and efficiency.

The support of network topology parameters and energy consumption efficiency induce additional complexity but are features that have become indispensable in modern architectures. The scientists can select upon best fit or first fit methods for the support of those features. Best fit approaches provide better methods for efficient system exploitation and can reduce internal fragmentation but in expense of bigger waiting times or possible external fragmentation. On the other hand first fit approaches may induce internal fragmentation avoiding external fragmentation and providing small waiting times with worst efficiency in system exploitation than the best fit. So depending the workloads and the platforms goals administrators need to scale well the trade-offs observed among the solutions.

The use of specific kind of jobs can particularly optimize the performance of the scheduler The array job (or bulk) is a specific kind of structures that allow a sequence of jobs that share the same executable and resource requirements, but have different input files, to be submitted, controlled, and monitored as a single unit. This allows the grouping of jobs which contributes in removing a big scheduling and launching overhead. Therefore only one schedule is performed for the array job in place of scheduling a big number of individual jobs. Furthermore the use of flexible jobs like moldable or malleable can optimize the schedule because they can adapt to system's availabilities [111].

The use of advanced reservations upon job scheduling can have a large negative impact on slowdown (stretch time) and average queue wait time but not necessarily on system utilization [113]. However this is very dependent on the workload. In general the less reservations are injected, the better for an overall system utilization. A good tradeoff can be attained by setting a limit for maximum reservations per user or day.

In general post-treatment workload studies of a specific system is of particular importance because it can reveal scenarios that could not have been realized differently [123]. A change on the

scheduling policy or a specific parameter may provide an important optimization. Scientists need to deal with a big number of tradeoffs for all different cases and there is never one best solution to be applied upon all contexts. Particular experimental approaches like the dynP scheduler [124] 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. On the same context of self-tuning scheduling, the work in [125] describes an approach based on flexible coscheduling that can automatically adapt the scheduling of jobs according to dynamic measurement of applications' communication patterns and proper synchronization needs of each application and trying to optimize the entire system's performance while addressing these needs.

3.4 Survey of Resource and Job Management Systems

In this section we present some of the most common Resource and Job Management Systems of our time. We provide an analysis of each one of them according to the main points discussed on the previous sections: a) Principal Concepts and b) Approaches to deal with Research challenges. Annexe 1 provides analytical tables with each systems supported functionalities for conceptual comparison. Finally, in section 5 we provide an overall synthesis and discussion upon all systems.

3.4.1 Commercial RJMS

The commercial Resource and Job management systems provide highly developed solutions and better maintenance support when compared to their open-source opponents. Their main advantages are their support of a big number of architecture platforms and operating systems, their highly developed graphic interface for visualization, monitoring and transparency of use along with a good support for Interfacing standards like Parallel libraries, Grids and Clouds. Moreover the software is normally installed and maintened by specialized expert teams.

Loadleveler

LoadLeveler [126] is a RJMS production system developed by IBM and initially based on CONDOR open-source system. It is a commercial IBM product designed initially for IBM clusters but it can also be used in any cluster. It works with AIX and LINUX operating systems.

LoadLeveler is mainly a Resource and Job Manager that offers basic functionalities as scheduler. Its architecture [127] is designed with scalability as the main concern. It is composed by several daemons running upon the master and execution hosts. The central controller called negotiator is responsible for collecting resource information, keep system wide objects and perform general coordination of daemons and requests. The scheduler daemons can be configured on multiple machines. One of the central differences is that the system's wide tasks such as scheduling and collecting information from execution hosts are distributed to different daemons. These mechanisms ease the task of the negotiator and contributes to an overall scalable design.

LoadLeveler makes use of *stanzas* which are groups of machines along with their selected characteristics. Specialized daemons make the selection of the particular machine (machines) that the job will be assigned to. Its Resource Management subsystem supports basic functionalities like scalable job launching, task placement using the cpuset mechanism and advanced features like high availability where any execution host can become central negotiator in case of failure of the former. However LoadLeveler lacks the support of energy efficient techniques and network topology aware scheduling.

As job manager it uses *classes* which are classifications to which a job can belong. For example, short running jobs may belong to a job class called short jobs. The system administrator can define these job classes and select the users that are authorized to submit jobs of these classes. It is possible to specify which types of jobs will run on a machine by specifying the types of job classes the machine will support. Queuing of jobs is implemented internally. Queues are list of jobs that are waiting to be processed upon each class.

When a user submits a job to LoadLeveler, the job is entered into an internal database, which resides on one of the machines in the LoadLeveler cluster, until it is ready to be dispatched to run on another machine. It provides accounting and reporting console and tight integration with some MPI libraries (MPICH and MVAPICH) but does not provide support for interfacing with Grid technologies, neither graphical or visualization interface.

Loadleveler's scheduler [128] supports backfill, preemption and fairsharing. It provides interfacing capabilities to integrate with external scheduler such as Maui, Moab or Catalina and an API to implement new scheduler like EASY [114]. Loadleveler was the first RJMS to support Gang scheduling [129] but since version 3.4.3 it removed the support for this policy.

LSF

LSF is a commercial RJMS supported by Platform Computing, considered as one of the most evaluated job schedulers. This software is originated from Utopia system [130] which was a research project, developed in the university of Toronto, providing dynamic and transparent load sharing functionalities in large-scale distributed systems.

Since the beggining LSF has been evolving mostly as a job management and scheduling system and less as a resource management solution. The architecture of LSF [76] has a rather complicated design with several daemons running on the central controller and the computing nodes. 5 daemons function on the central controller responsible for collection of resource management information, initiation of communications with the computing nodes and scheduling, whereas 4 daemons continuously turn on the execution hosts for the correct job execution and control. The controller-node and intra-node communication is made by sockets and remote procedure calls. There is a possibility to use a database for reporting and accounting information.

Its Resource Management subsystem supports basic features like task placement, resources monitoring and high availability, but lacks features like hierarchical view of resources or network topology aware scheduling. Even if the resources monitoring and reporting console is evoluated enough with detailed information like CPU and I/O bandwidth utilization the resources treatment is static and the task placement affinity is set by the administrator.

In order to provide a complete RJMS solution vendors like HP and BULL used integrated technologies of LSF scheduler upon SLURM [131] or RMS resource managers. Nevertheless, some features like user or kernel level checkpoint restart and preemption were not supported. In any way the integration of Resource Manager with a Job Scheduler seems to be less used nowadays because of complicated designs and overheads due to the interaction of two different systems.

However, the LSF scheduler provides a lot of enhancements when compared to other job schedulers and it was the first scheduler to offer the possibility to configure one scheduling algorithm per queue. LSF scheduler makes use of queues for job dispatching. Queues implement different job scheduling and control policies. All jobs submitted to the same queue share the same scheduling and control policy. Each queue can use all server hosts, or a configured subset of the server hosts. There are several scheduling policies in LSF: the simple FCFS, Fairshare scheduling, Deadline and exclusive scheduling, preemptive scheduling, and SLA-driven scheduling. A very important functionality supported by LSF is the feature of malleable and evolving jobs. That means that jobs can dynamically change their allocation, either per system or per application call, during their execution. For evolving jobs it provides only the possibility of shrinking. In addition LSF provides tight

integration with all MPI libraries and has interfacing facilities with Grid standards like DRMAA.

There are some innovative features concerning task placement and energy consumption efficiency. For the first one a job can be assigned a specific amount of memory which provides an important advantage for multicore architectures [132]. Furthermore, studies for understanding and predicting thermodynamics of the cluster have been initiated and jobs could be intelligently placed upon 'cooler' machines in order to maintain or reduce the overall temperature [133].

Finally LSF has been reported to be scalable upto 40000+ CPUs reaching the limits of 100000 jobs scheduling per day LSF provides an opensource limited version of their scheduler which called LAVA. It supports limited features of the commercial full version and it is scalable until 512 nodes. It is currently integrated as one of the schedulers of NPACI Rocks Cluster Solution [134].

Moab

Moab [135] is a job scheduler originated of PBS resource and job management system. It is the commercial version of the widely used Maui scheduler [136],[137],[138]. It does not provide a resource and job management subsystem so it has to be integrated upon a specialized software like Torque, LoadLeveler, LSF, SLURM or RMS. Moab enables capabilities through simply directing the resource and job manager regarding when, where, and how to run and manipulate jobs.

This integration implies that the queuing management is made on the resource and job manager side, thus Moab default configuration is to use a single global queue, however it may be configured to use multiple queues internally if needed. It provides the concept of throttling policies which can be imposed to define resource limits on users, groups, and accounts of interest as well as elaborate job prioritization control to allow jobs to be ordered according to credentials, resources, history, desired service levels and other attributes.

The integration with the Resource and Job Manager allows the interchange of all kind of information that may conclude in efficient scheduling. The concept of node sets has been adopted in order to improve the performance of parallel jobs upon heterogeneous platforms. In addition, the treatment of consumable resources like CPU, Memory, swap space and disk bandwidth contributes to best-fit task placement decisions. Nevertheless, this information interchange is the reason of important complicated integration design and overall overheads.

It provides all the basic and most of the enhanced features for job scheduling in HPC like a big number of scheduling policies (backfill, preemption, fairshare, multipriority-based), advanced reservations and license managers integration. Furthermore the support of moldable and malleable jobs is supported with the limitation that the underlying resource and job manager provides also

the necessary mechanisms for modifying the amount of resources of a job.

Scalability and efficiency issues have also been taken into account. The use of node sets allows the dynamic grouping of resources which contributes to a more scalable scheduling. The system was reported to scale good upto 160000 cpus and 40000 jobs.

PBSPro

PBS Pro [139] is a RJMS which was developed in NASA laboratories and came as an extension to the older Network Queuing System (NQS).

It is based upon a simple architectural design [140],[76] with support to a big number of architectures and operating systems. The central controller is running 2 daemons, one for answering requests and exchanging information with the execution hosts and the other one for scheduling purposes. The execution hosts maintain only one which is responsible for managing the physical resources, applying policies and task launching. The resource and job management subsystems are enhanced with most of the basic and advanced features. In particular, the concept of *vnodes* which is similar to the *node sets* described previously for Moab allow a finer resources grouping for a more efficient task placement. It provides enhanced topology aware scheduling and high availability techniques. The concept of cycle harvesting nodes as used in Condor also exists under PBSPro which allows the integration of desktop grid approaches upon PBSPro managed clusters. Moreover, it offers one of the most powerfull graphical interface with a web submission portal and visuazilization monitoring capabilities along with enhanced accounting and reporting tools with charts. Tight integration with Parallel libraries and interfacing with Globus is also proposed.

An interesting mechanism initiated in PBSPro is that of *hooks* which are objects that allow to accept or reject specific actions like modify input parameters and change internal or external values. The hooks can be executed before or during a job is run to provide specific modifications upon a job or a reservation.

As far as the scheduling subsystem concerns it supports most of the general policies in scheduling, such as FIFO, backfill, fairshare, preemption and multi-priority based. It allows one scheduling mechanism per queue and by default, jobs in the highest priority queues will be considered for execution before jobs from the next highest priority queue. This is configurable as a round-robin fashion that queues will be cycled through attempting to run one job from each queue. There are two types of queues, routing and execution. A routing queue is used to move jobs to other queues including those that exist on the different PBS servers.

One major drawback is that the accounting features use flat files for data storing in place of

a database. Apart the complicated design of storing information upon files in place of a database this can impose overheads on the efficiency of the fairsharing scheduling policy. Even if the programming structure is not known since the project is commercial it seems that it is not flexible enough.

3.4.2 Opensource RJMS

The opensource RJMS, initiated by universities and research centers, even if most of them do not support a big number of platforms and operating systems they provide more innovation and a certain flexibility when compared to the commercial solutions.

CONDOR

Condor [141] is the older and most innovative opensource Resource and Job Management System that is still in production. Developed by the Condor research project at the University of Wisconsin-Madison, it has been used as a research tool and a production system since 1984. It is a unique RJMS in the sense that since all other solutions aim for High Performance Computing, which means big amounts of computing power over a short period of time, Condor aims for High Throughput Computing (HTC) [142]. The goal of a High-Throughput Computing environment is to provide large amounts of computational power over prolonged periods of time by effectively utilizing all resources available to the network. The key to HTC is to efficiently harness the use of all available resources even individual workstations. Condor can manage a dedicated cluster as all other RJMS do. Its main advantage comes from the ability to effectively harness non-dedicated, preexisting resources under distributed ownership [143], [144]. This concept is called opportunistic computing.

One of its basic concepts is the expression of ClassAd. ClassAds are a flexible mechanism for representing the characteristics and constraints of resources, jobs, submitters or even daemons in the Condor system. It is a set of uniquely named attributes. They provide an expressive framework for matchmaking resource requests with resource offers.

As a Resource Manager [145] Condor supports cpu affinity techniques upon multicore and NUMA architectures. The way SMP machines are represented to the Condor system is that the shared resources are broken up into individual slots. Each slot can be matched and claimed by users and it is represented by individual ClassAd.

It's main strength and flexibility is centered on its job management system. It provides the support of different kind of environments, as called in Condor system, (standard, parallel, virtual

machines, java, etc) which depict the support of different kind of types of jobs along with specific quality of services for each one of them. Built-in Checkpoint/restart techniques, virtual machine deployment, enhanced security mechanisms, powerfull data-staging mechanisms are some of the services offered. Furthermore, a sophisticated mechanism exists to provide the description of inter-job dependencies along with an explicit meta-scheduler (DAGMAN), that is responsible for scheduling, recovering and reporting inter-dependent programs inside Condor. Condor offers the possibility for direct applications interfacing through a web service API using SOAP or DRMAA API. One of its main strong aspects is its support of every kind of grid computing approach either mainstream standards like Globus through Condor-G, alternative desktop grid approaches for bag-of-tasks applications like Boinc through Condor Flocking or even interfacing upon Clouds like Hadoop or Amazon EC.

The architecture of Condor's scheduler is different than the other RJMS. It is the result of cooperation of two different daemons. The first one (sched) maintains the submitted jobs in a queue and calculates job priorities, the second (negotiator) is responsible for making the matching of the job characteristics upon resources. The matching information are send back to the sched daemon which can make subsequent scheduling decisions before it launches the jobs. This hierarchical, distributed scheduling architecture makes Condor more scalable and flexible especially for multicluster platforms. Nevertheless, its scheduling power decreases when it comes for dedicated resources. Its scheduler is mainly governed by multi-priorities and fairsharing policies along with preemption features. But the major drawback is is that there is no backfill implementation for scheduling optimizations which can result into innefficient system exploitation.

Concerning Scalability and Efficiency issues Condor supports High Availability and Energy Efficiency, proposes internal node topology aware placement techniques and scalable sceduling for non dedicated resources. However, there is no network-topology aware scheduling. and its propagation techniques do not use any enhanced algorithm for optimization and since they are characterized by highly secure communication mechanisms they will have the drawback of poor scalability.

SGE

Sun Grid Engine (SGE) [146] which has recently became Oracle Grid Engine is an enhanced opensource RJMS migrated from CODINE which originated from DQS. Sun Grid Engine [147] aggregates the compute power available in dedicated computing clusters, networked servers and desktop workstations, and presents a single access point to users needing compute cycles.

Like PBSPro its architecture is based upon simple deamons exchanging information for job submission, scheduling decisions and task placement. The design makes use of a relational database for storing reporting information for nodes states and accounting information for jobs. A backup controller guarantees the high availability of the system and mechanisms for any computing node to become central controller in case of further failure of the backup controller exists. It provides a kind of hierarchical treatment of resources by defining concepts like host groups, and consumable resources. It allows a fine task placement which can bind tasks upon sockets,cores and threads.

Its job management system supports all basic type of jobs along with array jobs and job with dependencies. It supports only MPICH parallel library with tight-coupled integration. This is a major drawback for systems that use OpenMPI or MVAPICH libraries because those implementation are loosely coupled integrated. The disadvantages of loosely coupled integration are two folds, first the user needs to write specific scripts for each different MPI library, which may vary with their different programming model and second, the spawned processes escapes the control and monitoring of SGE after launching a job. In the case of loosely coupled integration there is no meaning of using enhanced task placement techniques like cpuset because processes will not be binded upon the specified cores.

On the other hand, a major advantage of SGE job manager is its support to a specific Grid standardized API called Distributed Resource Management Application API (DRMAA) that allows programs to interface directly upon SGE. Furthermore, a command line along with a graphical user interface is proposed. An advanced accounting and reporting console which makes use of the SQL database is also provided.

Finally the SGE default scheduling subsystem [148] follows a rather complicated design and provides differences with the other schedulers. It uses the concepts of *queues* as a logical abstraction that aggregates a set of job slots across one or more execution hosts. A queue also defines attributes related to priority policies and how jobs are going to be executed upon the hosts. The scheduler selects a job from a queue. In addition it uses the concept of *tickets* where each ticket policy has a ticket pool from which tickets are allocated to jobs that are entering the Grid Engine system. The queue from which the job is going to be selected for execution is chosen by the amount of tickets. Each routine ticket policy that is in force allocates some tickets to each new job. The ticket policy can reallocate tickets to the executing job at each scheduling interval. Only one scheduler can be used for the system and this supports algorithms like tickets priority based, fairshare, backfill and preemption. SGE also supports advanced reservations and applications licenses integration.

However it seems that the Scheduler API is not flexible enough to allow the easy programming of a new external scheduler [76] and the complicated design does not seem easy enough for maintaining and scaling upto big number of nodes and jobs.

Nevertheless, innovative features exist like the slotwise preemption which is a bestfit preemption policy and provides the means to ensure that high priority jobs get the resources they need, while at the same time low priority jobs on the same host are not unnecessarily preempted, thus maximizing the host utilization.

Concerning the scalability and efficiency issues SGE provides methods for energy efficient consumption and its scheduler which runs as a thread in the central controller daemon, may enable faster job starts and improved job throughput. It seems that on the latest version there have been made enhancements to the communications protocols and the architectural design. Sun Grid Engine has been reported to operate clusters that have more 60,000 CPUs effectively. Even if it provides enhanced internal node topology placement techniques it does not support network topology aware scheduling.

Torque and Maui

Torque [149] is the opensource version of PBSPro commercial product while Maui [136] is the relevant opensource version of Moab. The maintenance of Maui has stopped since 2003 and all later enhancements for this product have passed to Moab. Torque and Maui [150] are tightly integrated and provide a complete solution that is widely used upon systems that prefer an opensource RJMS.

The architectural designs are the same with their commercial relevants and the only differences are limited to their supported features.

The Resource Manager subsystem provides high availability and internal node topology scheduling but it offers a limited resources treatment with no particular concern for multicore architectures. The support of cpuset enhances its task placement capabilities. A feature that usually lacks on other opensource RJMS and is provided in Torque is the support of file stagein and stegeout mechanisms. This feature is used by jobs that have need for data propagation before the start and after the end of a job. However this feature can be rather contrainting for scheduling efficiency when having a large number of nodes.

Maui scheduler supports different scheduling policies like FIFO, backfilling, fairsharing and preemption along with advanced reservations. No support for licenses managers or moldable and malleable jobs is proposed.

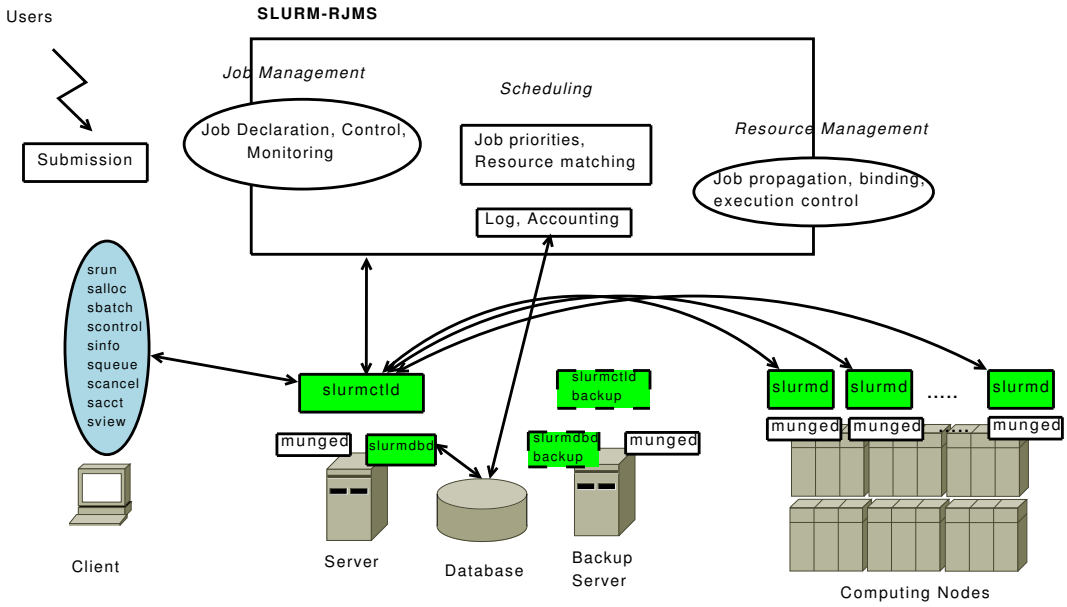


Figure 3.6: SLURM architecture

3.4.3 SLURM

SLURM [151] is an open source Resource and Job Management System designed for clusters of all sizes. In contrary with most of commercial and open-source RJMS it is developed to support only Linux platforms. It is one of the youngest and most evoluated open-source solutions and it is currently used upon some of the biggest computing clusters in the world according to TOP500.

SLURM initially provided more evolutions on the Resource Management layer giving particular care upon robustness, security, task placement and scalability, letting aside some important job management and scheduling issues. Thus, the use of an external scheduler was necessary if the users needed more advanced scheduling policies. Nevertheless, after some major enhancements upon the job management and scheduling layers during the last 2 years, the software can be used as a standalone system providing a lot of the advanced functionalities.

As shown in figure 3.6 SLURM's architecture follows the same design as other RJMS. It is based upon one central controller daemon upon the master node (slurmctld) which is responsible for the controls and procedures related to jobs and resources, a daemon responsible for the database controls (slurmdbd) and one daemon installed upon each computing node (slurmd) for information exchange with the master, task placement and resources monitoring. Specific daemons (munged)

are used to provide secure authentication. SLURM uses a general purpose plug-in mechanism to select various features such as scheduling policies, process tracking or node allocation mechanisms which allows it to be flexible and utilize custom as well as default methods. It uses the concept of *partitions* which represent group of nodes with specific characteristics such as job limits and access controls and it arbitrates conflicting requests for resources by managing one *queue* of pending work. The submitted user jobs can contain specific *job steps* which are are sets of (possibly parallel) tasks within a job that can utilize all or parts of the job's allocation. The concept of job step has similarities with the *array jobs* found on other RJMS and also with the *container jobs* found in OAR RJMS.

SLURM Resource Management

SLURM Resource Manager subsystem is characterized by its scalability and robustness. There is no hierarchical viewing of resources but there are specific plugins (select/cons_res) that can treat the resources upto finer granularities (like cores or threads). The concept of *partitions* can be used to treat the heterogeneity of clusters by grouping nodes with similar characteristics on the same partition. The communication between the master and the computing nodes is made with sockets and is normally based upon Ethernet Network. A common configuration mainly upon large clusters is to use Ethernet for SLURM communications and a second possible high-speed network connection like Infiniband or Myrinet exclusively for the applications communication. Furthermore, SLURM uses a sophisticated propagation algorithm based upon a dynamic adapted tree structure which has prooven to be highly scalable. SLURM provides specialized mechanisms for task placement according to the desired level of granularity and specific commands for resources monitoring and execution control.

Concerning the advanced resource management features SLURM provides secure communication with different ways to be configured: munged or authd authentication mechanisms with munged being the default method. A prototype using Kerberos exist as well. It provides high availability features by using a backup central controller (slurmctld_backup) and if a database is used for accounting purposes then a backup database daemon can be also used (slurmdbd_backup). A backup daemon is notified when a primary daemon fails and then the backup takes the control. The particularity of the approach is hidden on the database fault-tolerance technique, where SLURM uses a flat file to store all the necessary data while the database is down. When the database returns back to service the data are copied into the database and the normal function is restored. A specific overhead should be expected especially when the down-time was large and the amount of data to

be copied is significant.

The task placement in SLURM benefits of both network and hardware topology aware techniques which can result into important application performance improvements. Concerning hardware topology it provides two different plugins (select/linear or select/cons_res). The first manages jobs placement considering nodes to be one single resource for exclusive use whereas the second one takes into account finer granularities like socket, core and thread. The first approach is far more scalable than the second, but the second is indispensable for use upon SMP or NUMA architectures in order to avoid *internal fragmentation*. However, the second approach turns the launching and scheduling procedures much more complex, especially on large clusters, since they have to deal with much larger sets of resources. As described on section 3.5 the task placement upon resources is not enough and explicit binding of processes upon the cores or threads is needed for better treatment, control and performance. SLURM provides a specialized plugin (task/affinity) in order to deal with this issue. It supports both Linux kernel mechanisms `sched` and `cpuset`, with `sched` being the default. In addition a prototype implementation for support of `cgroups` exist already but a significant performance issue upon memory subsystem preventing it's official use for the moment.

Finally, SLURM provides energy reduction mechanisms by explicitly shuting down otherwise un-utilized machines after a predefined period of un-utilization. When jobs that need those nodes appear the nodes are waked-up again. The mechanism avoids to shut-down or wake-up big number of machines on the same time because of big electrical power imbalances hence this procedure is issued by chunks of machines.

SLURM Job Management

As Job Management System SLURM supports only simple and parallel jobs. However the support upon parallel jobs includes most of the known MPI implementations (MPICH, MVAPICH, OpenMPI, HP-MPI, ...) with tightly coupled integration. This means that SLURM initiates and controls the parallel job through the whole MPI execution. Some MPI implementations like BlueGene or Intel MPI are currently loosely coopled integrated initiated by ssh. The notion of array jobs which is a feature usefull for issuing big numbers of jobs with common characteristics is supported through the concept of job steps. Concerning security, job step digital signature verification can be used with munged or openSSL credential keys in order to authenticate the job step user upon an allocated computing node. These keys are the first messages to be propagated on job execution.

It provides various options for controlling the jobs before and after the start of execution. An important feature that exists only on LSF is the automatic shrinking of job size during its execution. This is a limited support of evolving jobs. Interfaces for monitoring and controling jobs through command line or a graphical GUI (sview) exist but no explicit support for DRMAA API or any kind of grid or cloud computing interface is supported.

As far as the advanced features concerns, job fault-tolerance is offered through the support of BLCR system-level checkpoint/restart library. Application-level checkpoint/restart can be also provided to jobs through the use of signals. Suspend/Resume features using job or partition priorities along with accounting with enhanced features is provided. Finally debuggers like TotalView are also supported.

SLURM Scheduling

The until recently minimal scheduling subsystem of SLURM, defined by a simple FIFO policy and integration mechanisms for interfacing with external schedulers (like Moab, LSF, Maui or Catalina) has been recently replaced with an updated system which supports most of the known scheduling algorithms like backfill scheduling, fairsharing, preemption based on jobs or queues policies, gang scheduling, resource limits by user account, and sophisticated multifactor job prioritization algorithms. It is the only production RJMS that provides gang scheduling as an option for algorithms. This policy minimizes the jobs waiting times and under certain circumstances can contribute to better system exploitation and optimized jobs turnaround times. This is because it enables the efficient filling up of all the 'holes' in the scheduling space. It allows jobs to time-share their allocated resources and suspend/resume their execution for dedicated utilization under specific time intervals. However in case the application is large enough and cannot be suspended upon RAM it has to use the swap space of disk which will lead to bad scheduling performance. In SLURM this scheduling policy is implemented with an option that defines the number of jobs that can share a resource.

SLURM provides one queue for pending jobs but the role of partitions share a lot of similarities with the queues defined in other RJMS like Torque or LSF. SLURM allows the use of only one scheduling algorithm per system. This is an important missing feature of SLURM and could provide a possible enhancement for future releases. Finally, some other existing features are the advanced reservation and the support for application licenses.

As far as the scalability features concern SLURM provides major enhancements and our experimentation results show better this argument.

SLURM Scalability and Efficiency

SLURM has been designed with scalability as one of its principal goals. It is currently the Resource and Job Management System of about the 40% of worlds largest computers according to TOP500 including a new system constructed by BULL for CEA research center called Tera100 which has a theoretic power of 1.25 Petaflops. SLURM systems' architectural choices based upon the precompiled language C and particular daemons: one for the central controller and one per computing node keep the system simple and highly scalable to the size of the clusters and responsive to large workloads. This scalability is also enhanced with the optimized propagation techniques for communication between the central controller and the nodes.

Concerning the efficiency issues, SLURM provides various scheduling policies like backfill,fairsharing and preemption and is the only RJMS that supports gang scheduling. Furthermore, optional best-fit algorithms for the topology aware placement of tasks, are provided as enhancement of the above policies, which can enable the scheduling of applications considering the particular architectural characteristics of the cluster. The only drawback that we can note is the fact that only one scheduler is supported at a time which could provide a bottleneck of the system if a particular scheduler encounters difficulties under particular workloads.

Experimental evaluations put more light on our conceptual observations in the last section of this chapter.

3.4.4 OAR

OAR [60] is an open source Resource and Job Management System. Similarly with SLURM it supports only Linux and MacOS platforms. It has been designed and implemented in LIG laboratory and it has been widely used as a research tool (CIMENT) and a production system (Grid5000) in various contexts. It is the default RJMS for one of the biggest international-wide computer science research platforms Grid5000 [152].

It is a rather particular RJMS that provides a lot of differences in architecture and concepts than all other solutions. Its uniqueness is hidden on its capacities for adaptability upon different kind of environments and its flexibility to adopt needed features easily.

OAR is based upon a particular modular architecture build around a relational database. An important characteristic of its architecture plays the programming language; which has been chosen to be mainly the high-level scripting language Perl (along with some modules in ruby and bash scripts). Its the only RJMS implemented solely on high-level languages and this choice is

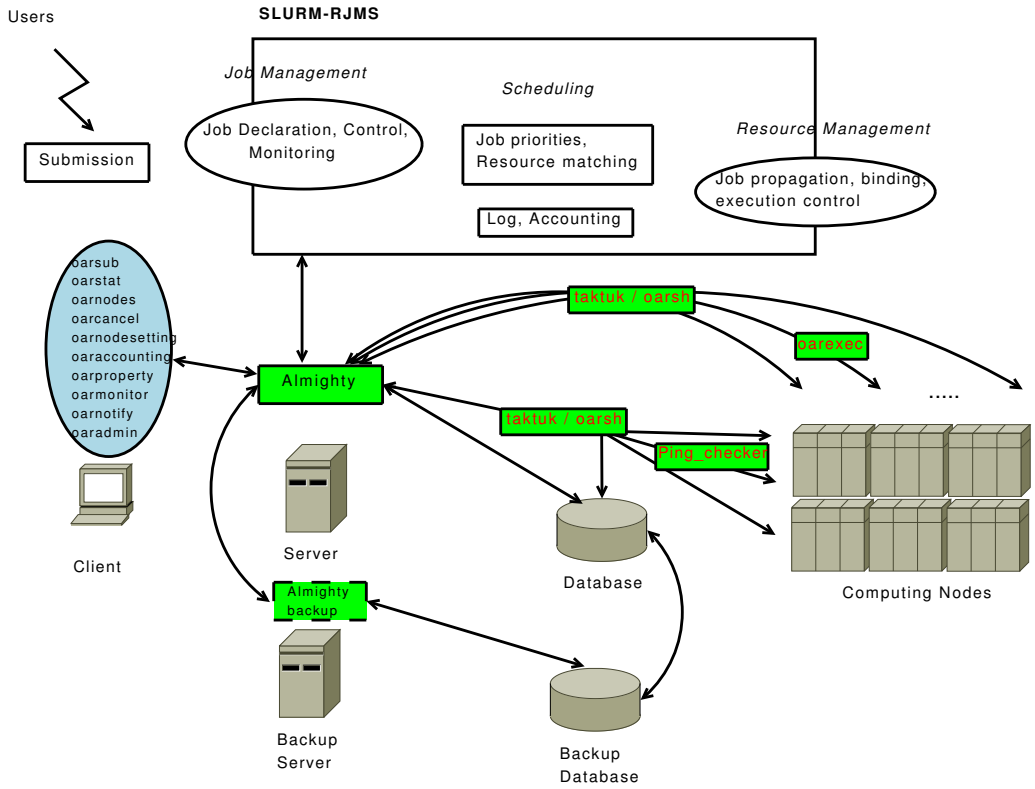


Figure 3.7: OAR architecture

motivated by two reasons : a scripting language is generally well suited for all the low-level system tasks (such as the distant execution of jobs on the nodes of the cluster) and it is fairly easy to develop simple programs using Perl because it has built-in high level data structures. However, due to the modular conception of OAR, it is possible to develop any part of the system in another language.

As we can notice on figure 3.7 the modular architecture of OAR is made of three main components : a database engine (MySQL or PostgreSQL), a central controller part *Almighty* (with an optional backup for high availability) and an 'on-the-fly' execution part with no daemons installed upon the computing nodes. All necessary commands for job launching (oarexec) and resources monitoring (Ping_checker) are made through special scripts that are propagated upon the nodes. The database engine is used to match ressources (using the rich expressive power of sql queries) and to store and exploit logging and accounting information. The interpreted scripting language

Perl, that is used for the modules implementation, has a straightforward syntax with built-in high-level data structures such as hash tables and regular expressions which make the development cycle short and the code both simple and concise. Perl is compiled on-the-fly during the execution of the script, so an overhead is expected compared to compiled languages such as C. The executive part of OAR is made of a collection of independent modules. Each of them is in charge of a small specific task. For instance, tasks such as jobs monitoring, jobs deletion, jobs submission, jobs execution, jobs scheduling, errors logging are all handled by separate modules. All these modules are executed each time the according task has to be performed. They all interact with the system using the database, which takes a central place within the architecture of the system. Contrary to other systems such as SLURM, LSF or Condor that also make use of an internal database engine, in OAR the use of the database is not limited to backup and accounting purposes, but hold all the internal data and thus is the only communication medium between modules.

OAR Resource Management

Based upon the above architecture the Resource Management subsystem of OAR provides an hierarchical view and treatment of resources. This means that a cluster is treated like a tree of resources where each resource may have parents and childs other resources depending on the architecture of the system. For example a switch will be connected upon nodes which will have cpus which in their turn will be composed by multi-cores and even multi-threads. The concept of hierarchical treatment of resources has been initially conceived to answer to the recent multicore architectures and their hierarchical model. A graphic representation of this hierarchical treatment is presented on figure 3.8. OAR also supports multicluster infrastructures which means that multiple clusters with their hierarchical trees can be imagined as childs of one single OAR Resource Manager.. OAR then will be in charge of submitting jobs, monitoring and accounting more than one clusters following the same administrative rules.

This representation and treatment is a very flexible and open model for management of resources allowing the support of any kind of architectures along with the management of heterogeneous clusters. Moreover it provides a simple way for users to describe their demands for resources. On the other hand dealing the resources with their finer granularity induce the scheduler into additional complexity, since it has to deal with a bigger number of possible execution host candidates, especially on the new many-cores architectures.

OAR is based upon a specifically adapted version of ssh network protocol (oarsh) for secure

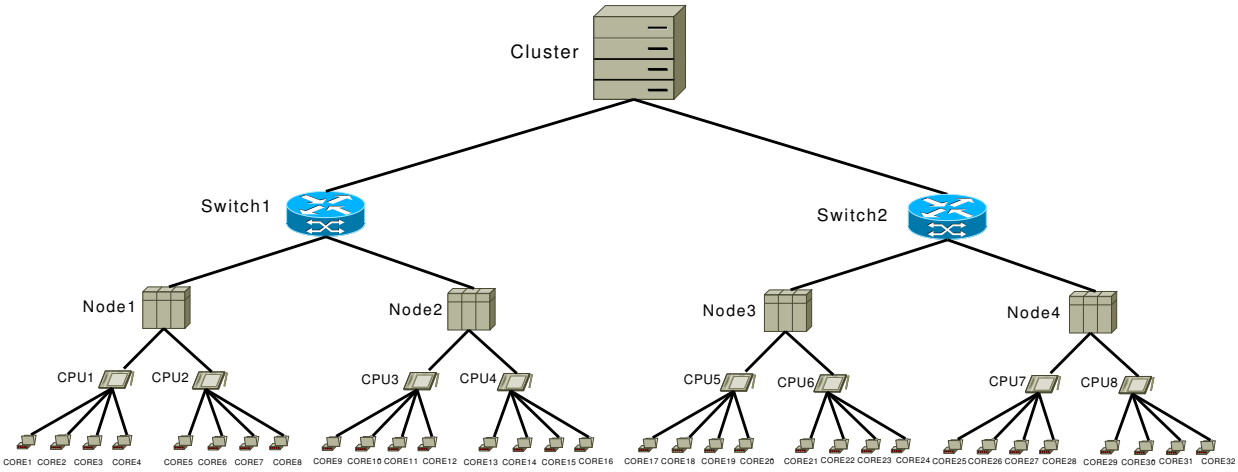


Figure 3.8: OAR hierarchical treatment of resources

communication between the server and the computing nodes and among the nodes. The authentication is made through ssh keys cryptography and a command (oardo) that allows users to execute programs with the security privileges of another user, similar function with sudo. The commands propagation and monitoring of tasks are handled by a separate tool (Taktuk [107]) that is called from OAR and interfaced with the database. Taktuk tool, as described in section 3.4.4 can use any kind of network protocol (like ssh or rsh) and in the case of OAR it uses oarsh protocol. Taktuk guarantees a scalable propagation and monitoring of resources for OAR.

Failure detection of nodes is made by testing their responsiveness to attempts for connection (reachability). Standard clients for remote execution have their own mechanisms to detect a failure on a single connection. These mechanisms rely on timeouts (during the response wait) and any node that is not reached by the time allowed for the initiation of the connection is considered as failed. As *Taktuk* uses an adaptative deployment tree, non responsive nodes do not take part in the deployment process. Thus, the duration of the failure detection last for the deployment time added to the timeout for the last connection. To improve the responsiveness and thereby the overall deployment time, timeouts for connection can be changed in *Taktuk*. This approach is the most flexible as it allows the user to choose the quality of service it needs: a very reactive behavior (with the risk of wrongly considering some nodes as failing) or a behavior closer to the actual nodes state (with a high confidence in failure detection but a low performance due to large timeouts).

Once the job is launched upon the computing nodes, the different processes need to be placed

upon the specific resources that were allocated to the job. OAR uses the cpuset (section 3.4.1) kernel mechanism in order to provide CPU affinity. This means that the job processes will be bound on the specific cores attributed to it. A problem that had to be solved was the case when a user wants to connect upon a specific node of his allocation in order to check details about the progress of his job. The specific processes of the interactive shell should be bound upon the created cpusets for his job allowing the user to use only the specific CPUS attributed to him. This was treated with the creation of oarsh which in reality is just a wrapper around the ssh command which through the propagation of environmental variables (OAR_CPUSET and OAR_JOB_USER) allows to a user to securely connect himself upon every node of his allocation and have the permission to use only his CPUS. This feature allows the proper management and cleaning of job processes after the end of the job.

OAR provides high availability mechanisms for the server and the database [153]. This functionality is based upon two different tools:1) the *heartbeat* which is a daemon responsible for monitoring services upon a server and in case of problem detection it can migrate the service on a backup server and 2)DRDB which is a fault-tolerance tool for databases which can mirror a whole block device via an assigned network. The functionality implies the use of one master server with a central database along with an additional server and database that will be used as backup in case of failure of the principal ones. The heartbeat daemons that allow the communication between both servers are enabled with a mechanism for providing the same IP to the clients through a mechansim called virtual-IP. Furthermore, DRDB guarantees the continuous mirroring of data, like a network based RAID-1.

OAR provides energy efficient management of resources through mechanisms that detect the machines unutilization periods and may trigger actions like `power-OFF` or `stand-BY`, in case of job appearance the machines are notified for *waking up*. As we will see on chapter 6 the energy reductions are traded-off with the waiting time of jobs while the machines are powered-ON. In order to decrease these waiting times, especially for jobs that demand small number of nodes, a technique that automatically keeps specific number of nodes *waken-UP* is provided as an enhancement.

Finally the topology aware placement is a feature that takes advantage of the powerfull hierarchical treatment of resources in OAR. This enables the management of resources as elements that belong to a specific branch of a tree. This can help for both network topology aware placement and internal node topologies characteristics. For the first case, this tree structure hierarchy allows a flexible selection of resources that belong to a specific number of islands through expressions that

can allocate particular branches of the tree. In a similar way the internal node topology placement can benefit of this hierarchical treatment since the tree it can go down until the threads in case of multithreaded architectures. The correct declaration of hardware internals matching the physical resources with the numerotation of the tree structure in OAR will allow an optimized selection of resources according to the application needs. This is enabled by the powerfull expressions provided by the OAR Job Management System.

The drawback of the OAR mechanisms for topology aware placement is that they make use of first-fit approaches of selection of resources which means that there is no specific algorithm that takes into account different choices between which there could exist a best-fit selection. This is an important difference with SLURM which provides best-fit algorithms for topology aware placement. Their mechanisms are compared in the last section of this chapter.

OAR Job Management

OAR has a powerful Job Management Subsystem characterized by the hierarchical viewing and treatment of resources and the support of different kind of jobs with ways to describe job attributes demands. The submission and monitoring of jobs in OAR is made through particular commands which are as separated as possible from the rest of the system. These commands send and retrieve information using directly the database and they interact with OAR modules by sending notifications to the central module. This direct use of database enables the expression of user demands properties through sql queries directly in the job descriptions, which gives a lot of flexibility to OAR jobs.

A particular concept used in OAR is that of admission rules. These rules are used to set the value of parameters that are not provided by the user and to check the validity of the submission. Possible parameters include a queue name, a limit on the execution time, the number of needed nodes and so on. The rules are stored as Perl code in the database and might be used to call an intermediate program so the admission can be as elaborate and general as needed. Admission rules can be as simple as setting a default duration of a job, or to make sure that a user does not ask too many resources at once; But also complex like providing automatic topology aware placement to the jobs tasks by selecting nodes of the same islands (section 3.5.2).

The hierarchical treatment of resources supported by OAR enables a simple and powerful way of expressing submission demands. It takes in consideration the topological architecture of a cluster and allows the user to express its demands with hierarchical manner in order to allocate specific branches of the hierarchical tree, structured by the resources. Hence, a hierarchical expression can

look like: switch=1/nodes=15/cpus=3/core=2 which demands 2 cores upon 3 sockets of 15 nodes, all located on the same island.

OAR supports a big number of types of jobs. Apart the standards (interactive, batch, parallel) it also supports:

- *array* job type with or without dependencies for the execution of jobs that use the same executables with different parameters,
- environment *deployment* job type through the integration with *kadeploy* tool [62, 63],
- *cosystem* job type which allow the encapsulation of other jobs in order to provide a virtual dedicated cluster for use by another RJMS,
- *moldable* job type provide a certain flexibility for the selection of the number of resources before the start of execution of the job with according execution times,
- *besteffort* job type which have the lower possible priority and always occupy otherwise unitilized resources with the compromise that they are forced for interruption in case a normal job demands the resources.
- *timesharing* job type provide a mechanism for allowing jobs to share the same resources with other timesharing jobs. This feature implies the use of the operating system scheduler for the sharing of the internal hardware resources (CPU, memory, I/O)
- *container* job type allow the submission of other `inner` OAR jobs inside them , like a sub-scheduling mechanism

It provides the possibility of application or system level checkpoint/restart through signals. However there is no implicit support of system-level checkpoint/restart mechanism like BLCR [154] or other. Techniques like suspend/resume are also implemented. The accounting of users jobs is provided with a straightforward manner through the database that keeps all the logistics of resources, events and workloads.

Concerning OAR systems' interfacing capabilities, the support of parallel libraries like MPI or PVM is provided through specialized scripts but there are no builtin parameters for the various implementations that exist. Nevertheless, this support is enhanced by tightly coupling mechanisms through the use of the particularly adapted for OAR ssh protocol called oarsh. Through this wrapper, OAR makes sure that the parallel tasks of an MPI job stay bound upon the particular resources that it has been granted access. Furthermore it can properly clean the resources after the end of

the job. There is no interfacing with Globus or Cloud approaches but there is integration with lightweight grid approaches like CIGRI (see chapter 4) which enables distributed large scale execution of jobs on the grid by aggregating idle resources of the clusters or ComputeMode which is a cycle-stealing software which allowing the construction of a virtual cluster through the use of independent PC's connected on the network.

In addition, there is no GUI or web portal for jobs submission but there are specific visualization interfaces for monitoring of jobs and a Restfull-API for development of exterior programs like web portal to interface with OAR. This API is based upon a cgi script being served by an http server (like Apache) that allows the programming of interfaces to OAR using a REST library. In this context most of Unix commands that exist for OAR can be implemented through http requests and this API based upon any programming language. REST defines a set of architectural principles by which Web services can be designed that focus on a system's resources. This includes how resource states are addressed and transferred over HTTP. REST has emerged in the last few years as a predominant Web service design model as a simple style of SOAP -and WSDL- based interfaces

OAR Scheduling

OAR scheduler is based upon the concepts of queues for general jobs grouping characterization. All the most important functionalities such as priorities on jobs, reservations, resources matching and backfilling are implemented. The priorities are managed through submission queues. All the jobs are submitted to a particular queue which has its own admission rules, scheduling policy and priority. A metascheduler exists to launch the schedule for each queue according to their priorities and provide management for advanced reservations. The scheduling module maintains an internal representation of the available ressources similar to a Gantt diagram and updates this diagram by removing time slots already reserved. Initially, the only occupied time slots are the ones on which some job is executing and the ones that have been reserved. The whole algorithm schedules each queue in turn by decreasing priority using it associated scheduler. An important advantage of OAR scheduling system is the capability of having a scheduling policy per queue and allowing the on-the-fly change of policy and queue through the metascheduler. Based upon the concept of Gantt diagram the internals of the scheduler enable mechanisms like simple estimation of the start time of a job or visualization of the scheduling decisions

Compared to approaches like Maui in which all the jobs are given an individual priority, OAR determines jobs priority using their queue. Of course both approaches are equivalent (it is sufficient to define a new queue for each distinct priority value) but queues make a partition of jobs into

groups. This is easier to handle for the administrator (an entire queue can be interrupted for some time or cancelled if needed) and this make possible different scheduling optimizations for different queues (response time for interactive jobs, throughput for large and slow computations, and so on). This represents a good tradeoff between simplicity and expression power and both the design and the understanding of the scheduler are extremely simple (policy for choice of queue and policy for choice of job in a queue).

As scheduling policies it currently supports only conservative backfilling, timesharing and fair-sharing. In addition the support of application licences is also provided.

OAR Scalability and Efficiency

The flexibility and modularity of OAR comes with a tradeoff in scalability issues. Indeed the architectural choices of OAR based upon the scripting programming language Perl which has to be compiled on-the-fly, and the hierarchical architecture which increases the number of elements that need to be treated by the scheduler provide important issues that need to be treated in order to make OAR efficient for large-scale computing clusters.

The development of a new prototype scheduler based upon CAML precompiled language along with optimizations to deal with the big number of elements in case of large clusters provide some initial steps towards optimizations of scalability. On the other hand the highly optimized propagation algorithms used for the exchange of information and submission of commands through the use of Taktuk provide a guarantee at this resource management level.

Moreover, we think that the ease of development cycle makes an appropriate approach for a research platform : as there is no interface to limit possibilities, scheduler developers can quickly implement prototypes for new functionalities, perform test and debug by accessing directly to the database. They also can easily make statistical or qualitative analysis on any internal data of the system using the sql engine.

OAR is a versatile resource and job management system easy for maintenance and support of new functionalities. That's why OAR has been used as our development testbed upon which we have implemented our prototypes for improving the system exploitation as we will see on the following chapters.

3.4.5 Synthesis

In this section we provide a general comparison of functionalities support among the above open-source and proprietary Resource and Job Management Systems. Table 3.3 presents a representative

summary of the evaluations and the extended version of the results are provided on the Annexes section 8.1 where each general concept of table 3.3 is decomposed into a number of particular features. The evaluation and comparison method is simple and it is based on assigning values to reflect the level of support of every concept by each RJMS. These values are presented on table 3.4. The extended version of the comparison (annexes, section 8.1) also includes an evaluation with points in order to obtain quantifiable results concerning the functionalities support of each RJMS.

Characteristics / RJMS Software	SLURM	CONDOR	TORQUE	OAR	SGE	MAUI	MOAB	LSF	PBSPro	LoadLeveler
RESOURCE MANAGEMENT FEATURES										
Resources Treatment	YY	YY	YY	YY	Y	NO	NO	YY	YY	YY
Job Launching, Propagation, Execution control	YYY	YY	YY	YY	YY	NO	NO	YYY	YYY	YY
Task Placement	YYY	YY	YY	YYY	YY	NO	NO	YY	YYY	YYY
JOB MANAGEMENT FEATURES										
Job Declaration	YY	YYY	YY	YY	YY	YY	YYY	YYY	YYY	YY
Job Control	YYY	YY	YY	YY	YY	YY	YY	YYY	YY	
Monitoring	YY	YY	YY	YY	YYY	YY	YYY	YYY	YYY	YYY
Authentication	YYY	YY	YY	YY	YY	Y	YY	YY	YYY	YYY
Quality of Services	YY	YY	YY	YY	YY	YY	YYY	YYY	YYY	YYY
Interfacing	YY	YYY	YY	YY	YYY	Y	YYY	YYY	YYY	YY
SCHEDULING FEATURES										
Scheduling Algorithms	YYY	YY	Y	YY	YY	YYY	YYY	YYY	YY	YYY
Queues Management	YY	YYY	YYY	YYY	YYY	YYY	YYY	YYY	YY	YY
SCALABILITY AND EFFICIENCY CHALLENGES										
Topology Aware Placement	YYY	YY	NO	Y	Y	NO	Y	Y	YY	YY
High Availability	YY	YY	NO	YY	YYY	Y	YYY	YY	YY	YYY
Energy Consumption	YY	YY	Y	YYY	YY	NO	YYY	YYY	YY	NO
Launcher and Scheduler	YYY	YY	Y	YY	YY	YY	YY	YYY	YY	YYY

Table 3.3: Conceptual comparison among various RJMS

Way/Level of Support	Representation
Advanced Support	YYY
Simple Support	YY
Limited Support	Y
No Support	NO

Table 3.4: Symbols used in the comparsion table

The overall quantifiable results of table 8.6 show that the RJMS which has the better noted functionalities support, overall, is the LSF system. The second is SLURM open-source system

and then OAR with SGE systems. With a closer look of independent evaluation results we can deduce that LSF has the most evoluated Job Management System with Condor being really close in notes, whereas SLURM has the best noted Resource Manager and Scheduling Subsystems. OAR provides a rather good Resource Management System whereas SGE has better notes in the Job Management layer.

However, we argue that our results concerning some concepts or features may not be 100/% correct because of limited documentations (especially for proprietary RJMS cases) or possible misleading understanding of the description of a feature in the provided documentations. Hence system like Loadleveler might be actually more evoluted than the notes reflect and this is because we think that not all of the details of the proprietary systems are published. Moreover, in the case of Maui and Moab the low overall notes are expected because those systems are meant to be integrated upon a Resource Manager System so their real evaluation should be centered on Job Management and Scheduling Subsystems. In a similar way, Condor software is not like the other RJMS since its main power comes on points where the other RJMS are weaker: on the ability to effectively harness non-dedicated resources and it is focused on features concerning High Throughput Computing. In our case, the choice of the particular features for comparison is made according to our point of views and tries to reflect the general current needs for Resource and Job Management Systems for High Performance Computing. Hence our results prove that Condor is a good RJMS for HPC and dedicated resources but fail to provide its strong points, since those features are out of the scope of our research.

Furthermore, the results concerning the general characteristics of table 8.1 have not been included in the final evaluation scores since they cannot be easily quantified. Nevertheless those results should give a precedence for commercial systems, Condor and SGE because they provide a bigger number of platforms and operating systems support, whereas SLURM, OAR and Torque are mainly developed for Linux type and MacOS systems. On the other hand, the type of particular programming language that has been used along with the number of code lines can provide a first insight of the complexity of the development of the software. In this case OAR system has the lighter approach of all systems with the less voluminous source code and this is one of the reasons of its high versatility and ease of development cycle.

3.5 Performance Evaluation of opensource RJMS

In this section we present and discuss real scale performance evaluation experiments of opensource Resource and Job Management Systems upon various cluster infrastructures. In the first part performance evaluation results concerning workload treatment efficiency with SLURM, OAR and Torque+Maui systems are provided. The second part presents results upon evaluation of network topology aware placement of SLURM and OAR systems. Finally 3.4.3 provides experiments concerning scalability issues of SLURM resource and job management system.

3.5.1 Launching and Scheduling Evaluation

In this first series of experiments our goal is to experiment with different opensource Resource and Job Management Systems and their capabilities to treat the same synthetic workloads and applications execution, deployed upon the same cluster infrastructures. For these tests we have made use of ESP benchmark [52] composed by a synthetic workload of 230 jobs with different sizes and target run times where each job executes the same parallel toy application `pchksum` (section 2.3.2). We have effectuated our experiments upon 2 different cluster infrastructures of Grid5000 platform.

- Genepi Cluster in Grenoble site with Intel Xeon E5420 QC 2.5 GHz biCPU-quadCores with 8GB of memory and network structured by 1Gigabit Ethernet + Infiniband 20G.
- Griffon Cluster in Nancy site with Intel Xeon L5420 2.5 Ghz biCPU-quadCores with 16 GB memory and network structured by 1Gigabit Ethernet + Infiniband 20G.

In these infrastructures the use of Infiniband is restricted to the applications MPI communication, which can benefit of the advantages of high-speed links, whereas Ethernet is used only for message exchanges for management purposes.

Regardless the real sizes of each cluster our experiments deployed personalized clusters, with different RJMS and policies for dedicated use, based on the methodology presented on chapter 2. The first set of deployments, made upon Genepi cluster, consisted of an 8 computing nodes system plus 1 separate node used for central server of the RJMS. The testbed consists of deploying OAR, SLURM and Torque+Maui as the responsible RJMS and observe their efficiency to treat the same synthetic workload (adapted ESP for 64resources cluster) under real dedicated conditions (no other users or jobs make use of the system). We have tested specifically selected scheduling policies of each RJMS. Each RJMS proposes different policies and various configurations (annexes), however

in this study we wanted to experiment with the most practical used ones. The final efficiency of each case is a function of multiple dependent internal procedures for each RJMS. However the main functions that are put under stress are the scheduling and launching capabilities of the RJMS.

RJMS/policy	OAR	SLURM	TORQUE+Maui
backfill	83.7%	83.9%	83.1%
preemption	Not Supported	84.9%	85.4%
gang-scheduling	Not Supported	94.8%	Not Supported

Table 3.5: OAR, SLURM and Torque+Maui experiments upon a cluster of 64 resources (8nodes-biCPU/quadCORE): Efficiency Percentage different scheduling policies for ESP benchmark

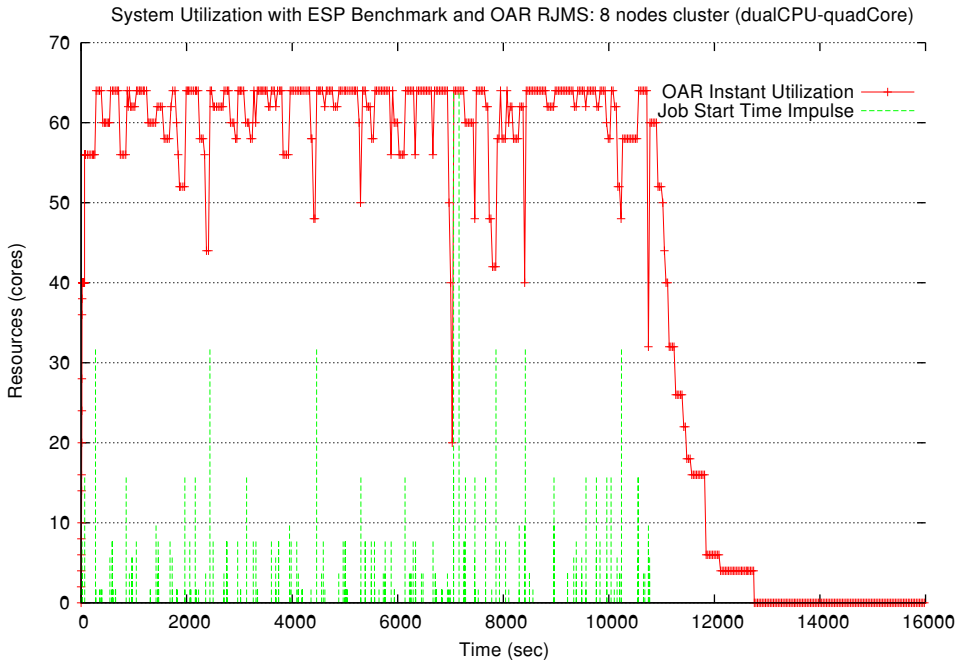


Figure 3.9: Instant system utilization for ESP Benchmark with OAR and Backfill policy (83.7% efficiency) upon a cluster of 64 resources

The results presented on table 3.5 show average values after 5 repetitions of the same experiment. The variations of results were rather small and the standard deviation s of the experiments

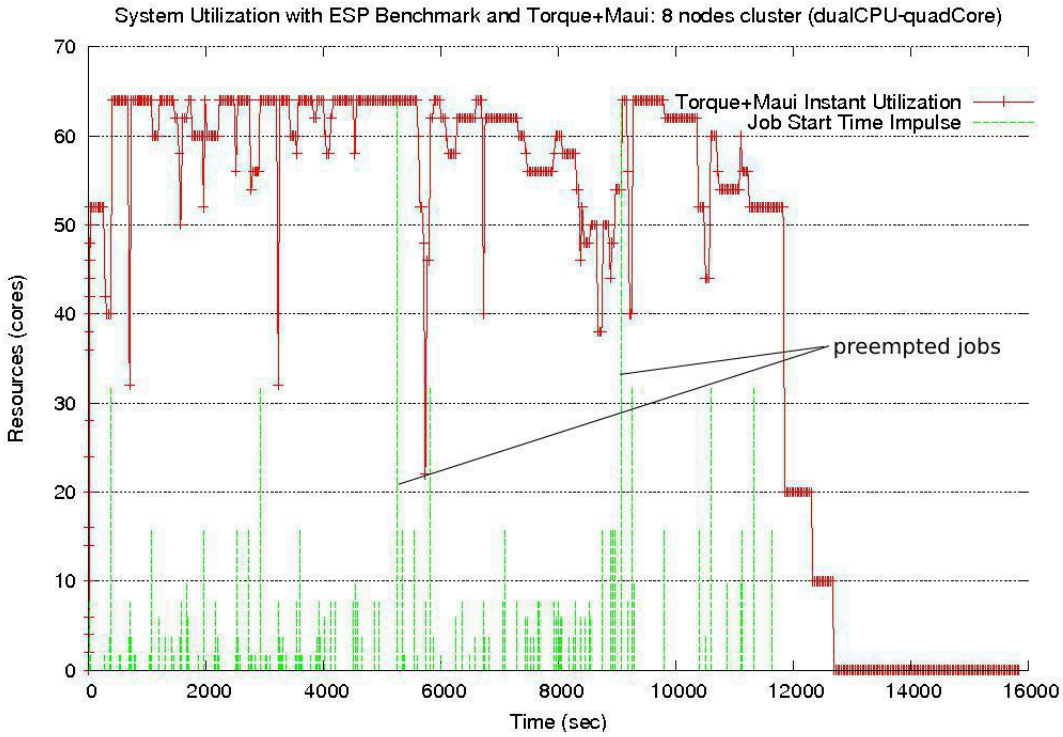


Figure 3.10: Instant system utilization for ESP Benchmark with Torque+Maui and Preemption policy (88.4% Efficiency) upon a cluster of 64 resources

remained with values below 2/% in all cases. We observe that backfill schedulers (conservative cases) of the tested RJMS have similar performances with a slight advantage of SLURM scheduler. We also see that preemption has much better results than simple backfill scheduling. This is because ESP benchmark makes use of 2 higher priority jobs which demand all the resources of the cluster. With normal backfill schedulers these 'all resources' jobs provide a barrier, constraining the backfilling of smaller jobs, during the time they are waiting for execution. On the other hand when preemption is used, these jobs are executed immediately so they do not constrain the backfilling.

The best performance is attained by gang-scheduling which is supported only by SLURM. The very good performance of this scheduling policy is related to the fact that it allows the efficient filling up of all the 'holes' in the scheduling space jobs. They make dedicated use of resources through suspend/resume techniques(as explained on section 3.3.4). The simpliciy of the particular application, used in ESP benchmark, eases the work of the system to perform suspension of

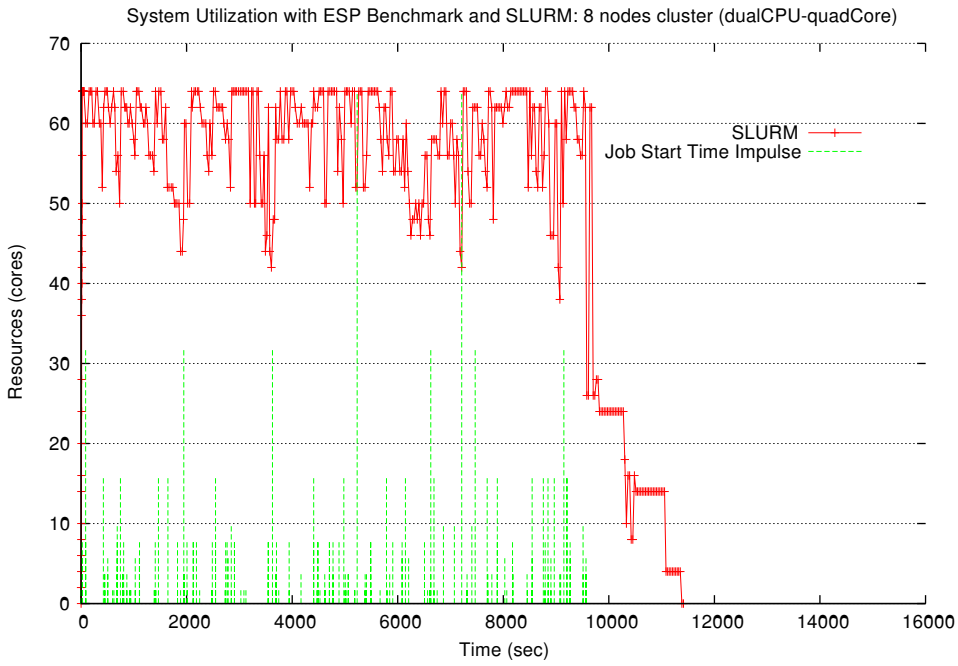


Figure 3.11: Instant system utilization for ESP Benchmark with SLURM and Gang-Scheduling policy (94.8% efficiency) upon a cluster of 64 resources

execution upon memory and no need for swapping is needed which would provide additional overhead. In particular, SLURM provides the possibility for configuration of 2,3 or 4 coscheduled jobs upon the same resource. In this experiment we have tested only with 1 pair of jobs upon the same resource.

Regardless the very good efficiency of gang scheduling, it is a policy that is not used in production. The reason is that in case the application cannot be suspended only upon RAM (which is the case for plenty of real applications) it has to use the swap space of disk and this will lead to additional performance degradation and scheduling overhead. On the other hand preemption, which makes use of the same technique of suspend/resume but only for selected high priority jobs; has become an indispensable feature that can really optimize the performance of the RJMS in production use. In this case the change of context is made only once so there is no big overhead upon scheduling. OAR RJMS needs to be enhanced with this policy as well.

Figures 3.9, 3.10, 3.11 show the instant system utilizations throughout the execution of the whole workload for different RJMS and scheduling policies. The figure provides the instant start time impulse of each job showing the size of the demanded resources. Only the behaviour of one

RJMS is shown for each policy since the graphs do not have big differences between them apart the final turnaround time of the whole workload execution. It is interesting to observe how the higher priority jobs are executed instantly when they are submitted in the cases of Torque+Maui preemption and SLURM gang-scheduling whereas they need to wait the termination of executing jobs and the scheduling decisions in the case of simple backfill case of OAR. The scheduler selected to delay the execution of the first Z-type job (that allocates all resources) and relate it with the second one (one after the other) as a best fit option. The same strategy is also followed by SLURM and Torque+Maui backfill schedulers

The second set of deployments was made upon Griffon cluster and consisted of a larger system of 64nodes (biCPU/quadCore) with a total of 512 computing resources and 1 central controller as the RJMS server. We performed the same experiments for backfill policies for each different RJMS case. Table 3.6 show average results after 5 repetitions of the experiments. It is interesting to observe how both OAR and Torque+Maui are induced to important performance degradation for ESP efficiencies.

RJMS/Results	OAR	SLURM	TORQUE+Maui
Average Wait time	3206sec	2872sec	3324sec
Total Execution time	13613sec	13013sec	13894
Efficieny for backfill policy	79.1%	82.7%	77.1%

Table 3.6: OAR, SLURM and Torque+Maui experiments upon a cluster of 512 resources (64nodes-biCPU/quadCORE): Efficiency Percentage for ESP benchmark and backfill policies

Closer observation of the internals of each system showed us that these degradations are due to different reasons for each case. OAR provides an hierarchical treatment of resources which facilitates their use and management but is induced to increased complexity with the scaling of the cluster size and the elements that need to be taken into account on the scheduling procedure. In addition the default scheduler is implemented in Perl scripting language which is less efficient due to the on-the-fly compilation of the code. New versions of the scheduler implemented in CAML which is powerfull pre-compiled language with advanced scheduling techniques are already prototyped and will be soon in production. Due to lack of time this new OAR scheduler has not been tested.

On the other side, for Torque+Maui system, the ESP efficiency degradation is not due to scheduling overheads but due to slower resource management and launching mechanisms. In

particular the fact that Torque resource manager does not provide resource management with fine granularities in relation with the use of CPUSET mechanism, for job confinement, still in experimental phase induced important overheads in the whole process. It has to be noted that this feature was configured in all 3 RJMS and both OAR and SLURM have provided its support much earlier than Torque. Experiments without the use of CPUSET shown better performance for Torque but still we believe that Torque needs to provide the option for finer granularities in order to reduce cases of internal fragmentation during scheduling. However newer versions of the system already provide the stable version of job confinement with CPUSET mechanisms.

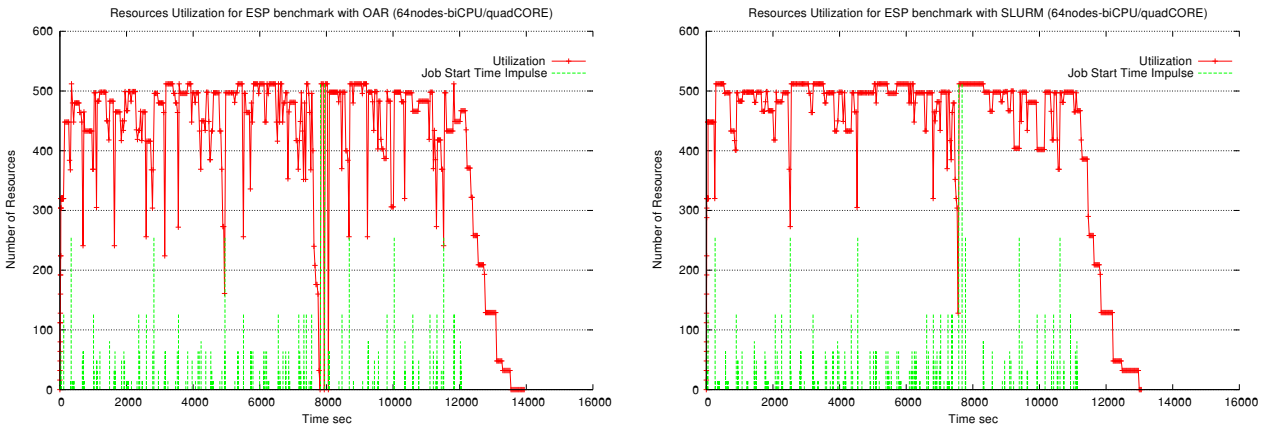


Figure 3.12: Instant system utilization for ESP Benchmark with OAR(79.1% Efficiency)(left) and SLURM(82.7% Efficiency)(right) upon a cluster of 512 resources (64nodes:biCPU-quadCORE)

Figures 3.12 show the instant system utilization along with the jobs start-time impulses for ESP execution with OAR and SLURM for the same experiments shown on table 3.6. The choices for job executions seem similar however SLURM manages to finish the whole workload faster than OAR. An interesting observation is to see how the management of SLURM leads to continuous constant system utilization whereas a lot of variations are observed for OAR case.

Figure 3.13 shows the cumulated distribution function on wait time for the same experiments. The advance of SLURM versus OAR is clear considering the impact of scheduling upon jobs waiting times. The same result can be also observed by the differnce in jobs average waiting time shown in table 3.6. In a whole, SLURM is more scalable than OAR and Torque+Maui and the scalability capabilities of SLURM will be also discussed on the following sections.

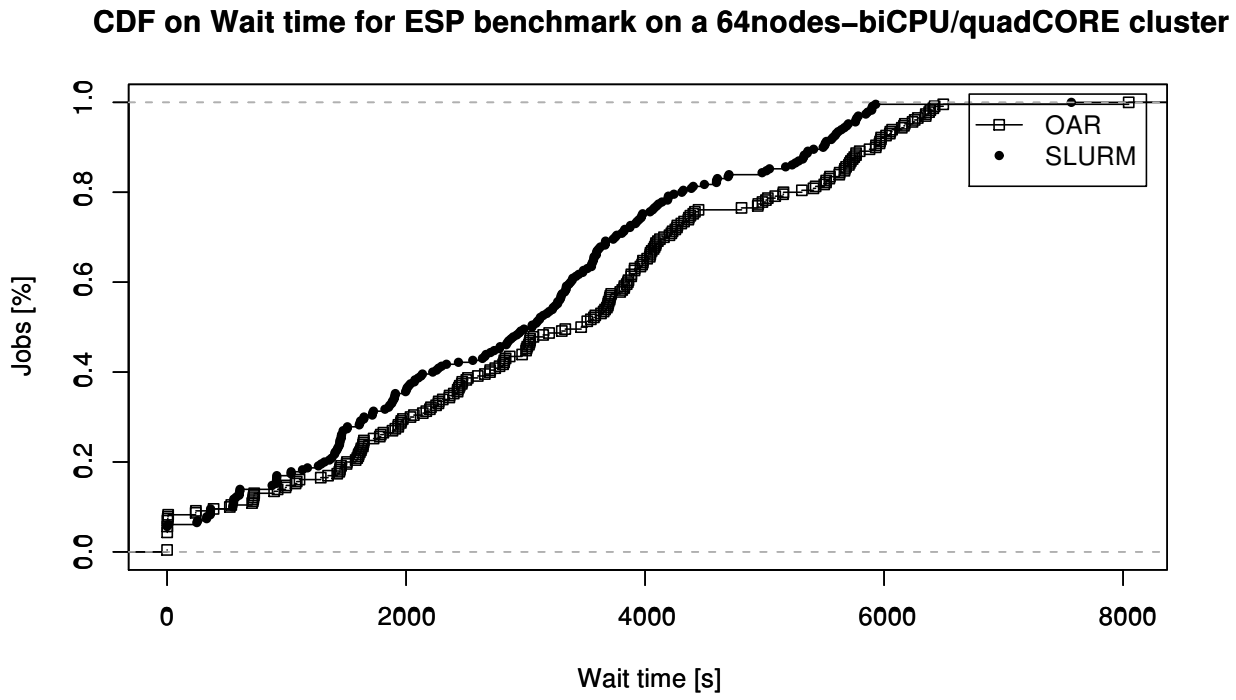


Figure 3.13: CDF on wait time for ESP benchmark and Backfill policies of OAR(79.1% Efficiency) and SLURM(82.7% Efficiency)

3.5.2 Network Topology Aware placement Evaluation

The goal of these second series of experiments is to evaluate the mechanisms for dealing with efficient placement of jobs regarding network topology characteristics. The experiments have been effectuated upon Griffon Cluster in Nancy site with Intel Xeon L5420 2.5 Ghz biCPU-quadCores with 16 GB memory and network structured by 1 Gigabit Ethernet + Infiniband 20G. For this suite of experiments we have taken into account the network topology architecture of the cluster. In particular we have chosen 64 nodes dispersed upon 3 different *islands* of nodes (groups of nodes that are connected upon the same switch 3.3.1) . At the particular cluster we have fat-tree network topology. Hence the performance degradation due to network contention is going to be inexistent. Our motivation is to experiment with the schedulers in order to obtain the best possible strategy for placement, thus we provide the mechanisms to favorize the network topology placement and we evaluate the efficiency depending on their choices for placing the jobs upon 1,2 or 3 islands.

For the experiments we have made use of the synthetic workload of ESP benchmark. The table

Job Type	Fraction of Job Size relative to total system size	Job size for a 512cores cluster (in cores)	Least number of needed islands	Count of the number of total jobs	Percentage of job instance	Target Run Time (Seconds)
A	0.03125	16	1	75	32.6%	267
B	0.06250	32	1	9	3.9%	322
C	0.50000	256	2	3	1.3%	534
D	0.25000	128	1	3	1.3%	616
E	0.50000	256	2	3	1.3%	315
F	0.06250	32	1	9	3.9%	1846
G	0.12500	64	1	6	2.6%	1334
H	0.15820	81	1	6	2.6%	1067
I	0.03125	16	1	24	10.4%	1432
J	0.06250	32	1	24	10.4%	725
K	0.09570	49	1	15	6.5%	487
L	0.12500	64	1	36	15.6%	366
M	0.25000	128	1	15	6.5%	187
Z	1.00000	512	3	2	0.9%	100
Total				230		

Table 3.7: ESP benchmark characteristics for 512 cores cluster

2.2 of the previous chapter has been adapted to fit the case of 512 cores cluster and the values are presented on table 3.7. The column *Least number of needed islands* shows that only 3 classes of jobs (C, E and Z representing the 2.6% of the total number of jobs) cannot be executed on 1 island (1 island contains 168 or 176 cores), the rest of them should be ideally placed upon 1 island.

The use of the default ESP benchmark implies that the target run-times of each job is fixed even if they make MPI communication between their allocated resources. This means that the topological placement capabilities of each RJMS (that could favor the execution times of specific applications) will not have any effect on the final efficiency of the RJMS. In contrast, efficient placement techniques may increase the waiting times of the jobs, since the scheduler will take more time to decide, which can lead to poor ESP efficiency.

In our evaluation we deal only with SLURM and OAR systems since Torque+Maui system does not provide mechanisms for dealing with network topology optimizations.

OAR through moldable jobs

The mechanism for dealing with network topology placement for OAR resource and job management system takes advantage of the hierarchical treatment of resources and the use of a specific kind of flexible jobs termed as moldable [60] (which adapt themselves according to availabilities before their execution). In OAR, every resource can be considered as a leaf with a different path in the hierarchical resources tree (figure 3.8). Hence the network topology architecture is known by default. This can be taken into account by OAR and jobs that can fit on single islands could be specifically placed upon one of them. According to this technique a normal job demanding a number of random resources can be modified to a moldable one that demands resources related to the islands that they are connected favoring those that are connected on a single island.

Different levels of flexibility can be provided through *moldable* jobs. The moldable jobs can be

adapted to allocate resources upon 1,2 or 3 islands according to the fastest availability or they can provide stricter options for a by default execution of jobs upon the exact switches that are needed.

Apart the choices that the user can make the administrator can provide by default network topology placement through OAR. For this a simple techinque which passes from the powerfull expression of admission rules exists. The administrator of the cluster can configure a specific admission rule to fit to the wanted strategy by automatically changing a normal job to moldable one.

SLURM through constraints or Topology aware plugin

In contrast with OAR, SLURM provides a specific plugin to support topology aware placement of jobs. The advantage of this plugin is its best-fit approach to deal with this aspect. This means that the plugin will not only favor the placement of jobs upon the number of switches that are really needed, but it will consider a best-fit approach for example to select a nearly full island in place of an empty one in order to leave the empty island for jobs that could ask more resources and can potetntially fit upon one island. Choices like this cannot be taken with the technique of moldable jobs of OAR which consider a first-fit approach. However since SLURM does not provide hierarchical management of resources the network topology architecture needs to be provided in a separate file where all the connections between the different levels of switches and the leaf nodes are provided. Besides the topology aware plugin similar functionality with moldable jobs is provided by the help of *constraints* which are features that have been assigned to resources by the administrator and can be used by the users in the form of parameters inside the jobs [151].

The administrator can specifically provide the different islands of nodes as features of each partition inside the main configuration file of SLURM. Similarly with the functionality of moldable jobs the constraints of user jobs can provide different levels of flexibility allowing the user to adapt its job according to the sensitivity of the application to network topology placement. Hence a large time-consuming application sensitive to network communication could be adapted for strict use of exactly the number of switches that it fits in, whereas smaller applications with less sensitivity could simply benefit of the automatic topology aware scheduling of the plugin.

Figure 3.14 shows the efficiency of the above placement strategies for the selection of resources that are contained on 1, 2 or 3 different islands of our cluster infrastructure for the different cases of OAR and SLURM. The barplots show the percentages of jobs that are placed upon 1, 2 or 3 islands during their execution with and without topology aware placement techniques. We can observe that the default hierarchical treatment of resources of OAR provides better placement for bigger percentage of jobs than the default placement of SLURM without considering topology

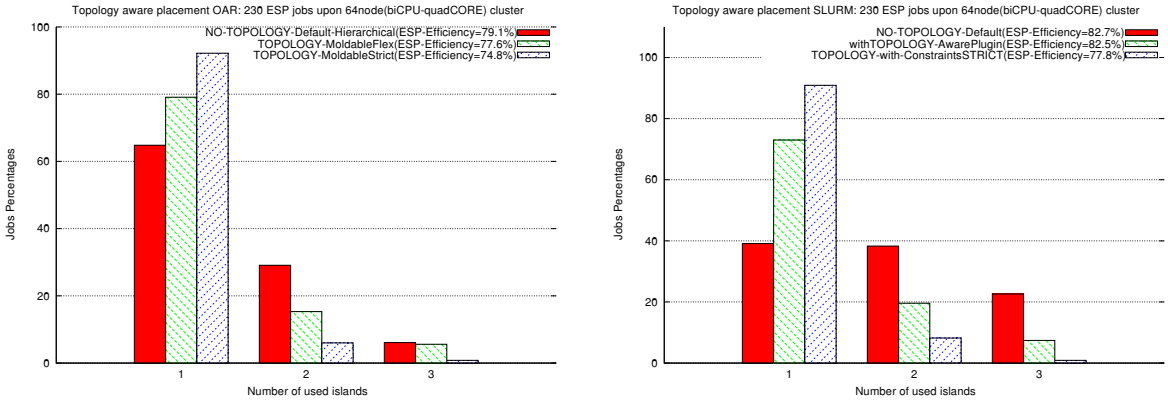


Figure 3.14: Efficiency of network Topology aware placement mechanisms OAR(left) vs SLURM(right)

ESP Characteristics	Theoretic Ideal values	SLURM with Topology aware plugin	OAR with Topology aware moldable jobs
Total Elapsed Time(sec)	10773	13050	13877
Average Waiting-Time(sec)	-	2987	3411
Efficiency	100%	82.5%	77.6%
Jobs on 1 island	222	168	182
Jobs on 2 islands	6	45	35
Jobs on 3 islands	2	17	13

Table 3.8: Topological and ESP characteristics with the default Topology aware placement techniques of SLURM and OAR

issues. This is an advance of the hierarchical treatment of resources which takes into account the different levels of granularities concerning the resources. On the other side this has an impact on the efficiency of the schedulers speed as wee can see on the efficiency of the ESP benchmark for each case. It is interesting to see how the topology aware plugin does not provide any impact on the efficiency of SLURM scheduler, since the ESP efficiency between the two cases is nearly the same, whereas an important improvement can be observed on the selection of islands. The topology aware plugin favors the placement regarding topology characteristics but if the best-fit choices cannot be attained then it compromises with not best-fit choice so as not to waste a lot of time due to external fragmentation.

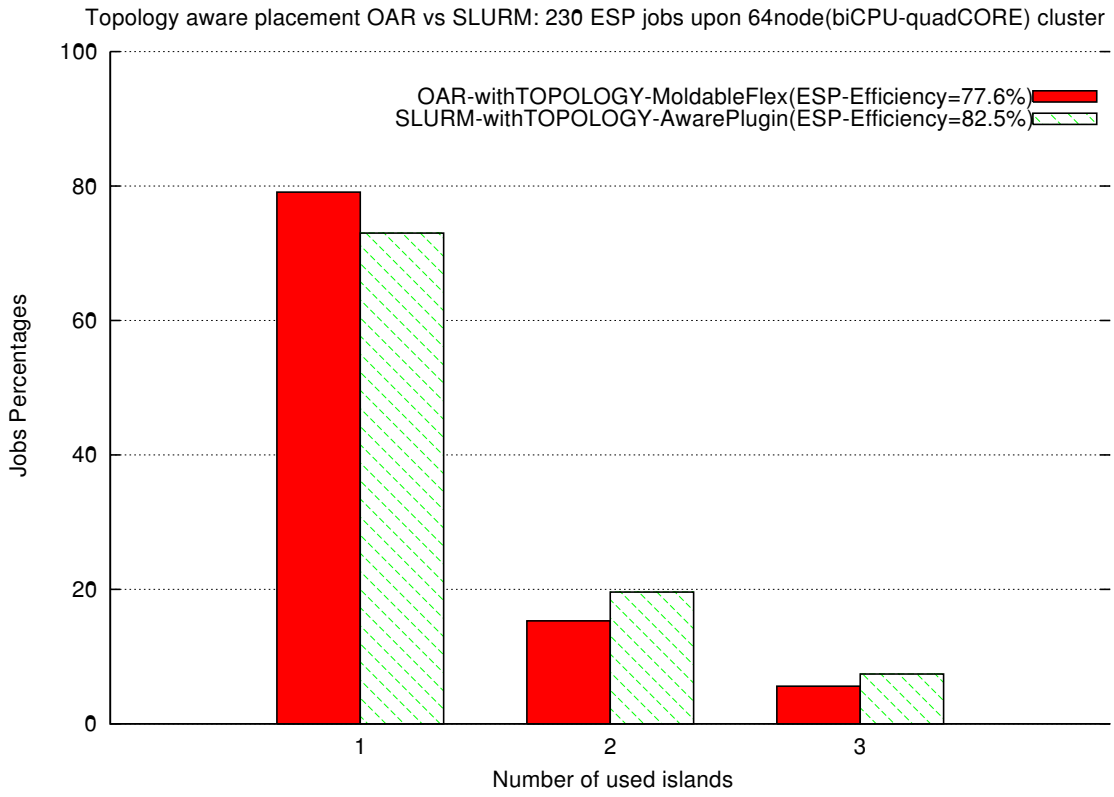


Figure 3.15: Efficiency of the Default Topology aware placement techniques for SLURM and OAR

We can see how the strict approaches of both cases present important degradation in performance due to larger waiting times for the best placement but with an impact on ESP-efficiency. Finally figures 3.15 and table 3.8 show in more detail the evaluation characteristics of the Default Topology aware placement techniques for SLURM and OAR. The results proove that OAR can attain better placement for more jobs with a loss in average waiting times and ESP efficiency. However SLURM can also attain similar results for better jobs placement with the use of the mechanism of constraints. In the same time OAR could attain similar efficiency with the use of faster scheduler.

3.5.3 Scalability and Efficiency Evaluation

In this third series of experiments we perform experiments of SLURM resource and job management system upon a dedicated cluster of large-size. The tests have been effectuated upon a

production BULL cluster constructed for testing purposes before it can be delivered to specific client in industry. This temporarary cluster was a part of the Tera100 supercomputer constructed by BULL for CEA-DAM research center. It is composed by Intel Xeon series 7500 processors (quadCPU-octoCORE) with 32GB of Memory and Ethernet+Infiniband networks. The Infiniband network is based on fat-tree topology with a pruning effect between the top level and the medium level switches. Additional architectural information cannot be given since they are BULL and CEA confidential.

For these experiments we have made use of the same workloads and applications that have been utilized on the previous chapters. In particular initially we used different cluster sizes (512 and 9216 resources) for adapted ESP benchmark and we observe the efficiency of the execution of whole ESP workload making use of the parallel toy application `pchksum`. Then we perform throughput experiments in order to measure the behaviour of SLURM under jobs submission burst. Finally a modified ESP benchmark has been introduced in order to provide a way to experiment with the actual efficiency of the topology aware placement mechanism of SLURM upon a fat-tree topology

Scaling size of cluster and size of jobs

Initially our goal is to check the efficiency of the scheduler and the launcher while scaling up the size of the cluster. For the specific experiments SLURM was configured with the following parameters:

- *task_affinity/CPUSETS*, for the task confinement upon particular number of processors(cores)),
- *cons_res*, for a fine granularity of task placement
- CR_Core_Memory, for task placement with granularities Core and Memory
- backfill scheduling policy, as permanent scheduling algorithm
- accounting with mysql + slurmdbd enabled, for jobs accounting with security
- SLURM-High Availability NOT enabled, which means that the backup control deamon and backup database deamon are not working
- Topology plugin NOT enabled, which means that the scheduling plugin that favors the placement of jobs upon the same or limited number of islands will not take effect
- Job priorities WITH Preemption, for an immediate execution of higher priority jobs after suspension of lower priority that occupy the demanded resources

Table 3.9 present the final evaluation results concerning the ESP benchmarks executions with 512 and 9216 cores. The degredation in efficiency is trivial and this represents a good result concerning the scalability of SLURM.

SLURM NB cores / ESP-Results	512	9216
Average Wait time(sec)	2766	2919
Total Execution time	12992	13099
Efficieny for backfill+preemption policy	82.9%	82.3%

Table 3.9: ESP benchmark results

Figures 3.16 show us the system utilization graph during the execution of the ESP synthetic workload for a configuration of 512 and 9216cores. In both figures the y-axxis represent the number of utilized resources and the x-axxis the time in seconds. We can observe that there are practically no differences in the scheduling of the tasks and that SLURM manages to keep a good system utilization while we increase the size of the cluster.

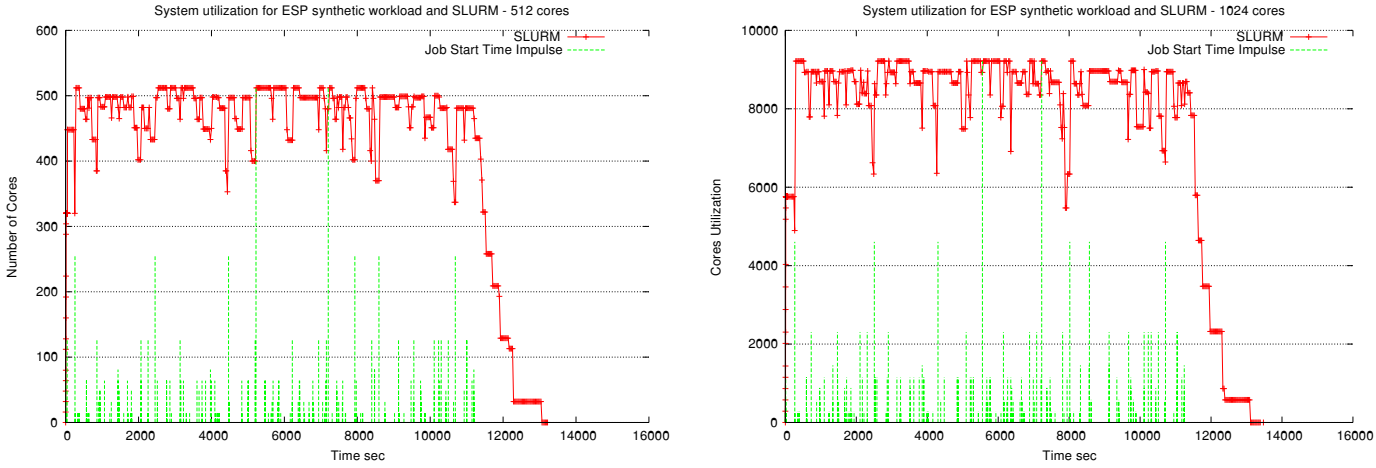


Figure 3.16: Instant system utilization for ESP Benchmark with SLURM and cluster size of 512 (left) and 9216 (right) resources

Figures 3.17 present the strech times of all the 230 jobs of the ESP workload for two cluster configurations of 512 and 9612cores. The less the factor for the jobs, better quality of service for the Resource and Job Management System. We can observe a slightly higher stretch on the last

jobs for the 9216resources case.

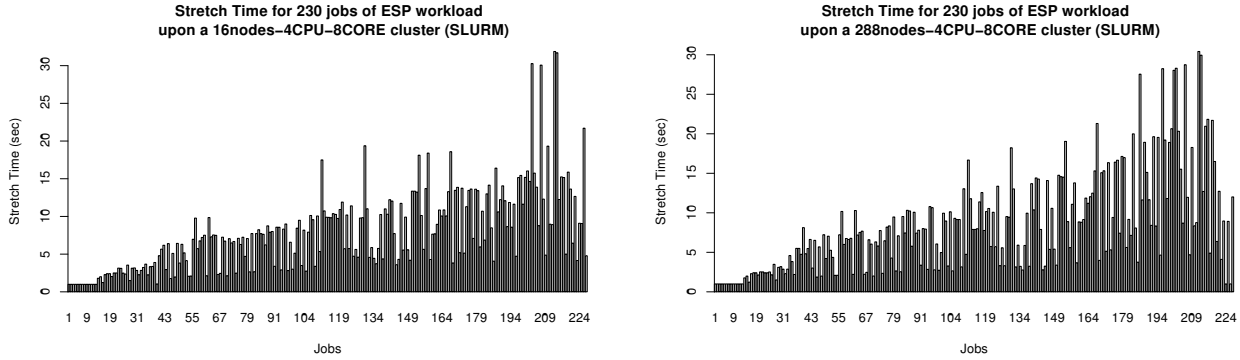


Figure 3.17: ESP Benchmark slowdown for SLURM 512-9216 cores

Figure 3.18 show us the cumulated distribution function of jobs waiting time during the execution of the ESP workload for the cases of 512 and 9216 cores. The graphs proove that the difference in waiting times are trivial as they are also shown by the average values of table 3.9.

Throughput experiments

The goal of these experiments is to stress the launching mechanisms of SLURM with large number of small sized jobs. In particular these experiments have been effectuated upon the same cluster as before with a configuration of 320 nodes for a total of 10240 resources. Our testbed is composed by a script that launches a for loop of batch jobs which demand for 1 core and perform a simple sleep function for 1000 seconds. Our testbed submitted 11000 jobs of this type in order to observe how many jobs are treated by SLURM per second and if there are any limits beyond which there is a degradation of performance.

The figures presented in 3.19 show the number of submitted jobs per second in the left and the number of terminated jobs per second in the right as treated by SLURM . In more detail the figure in the left implies the passage of jobs from Waiting to Running state whereas the figure in the right implies the passage of jobs from Running to Terminated state. Surprisingly the thoughput experiments of backfill scheduler showed us very good results with constant rate of about 160 jobs per second in average that start execution. Similarly in the right we observe the termination of jobs with constant rate of about 70 jobs per second in average which can arrive up to instant throughput of 220 jobs at the right of the graph.

We have effectuated the same experiments for backfill+preemption scheduling policy (with no

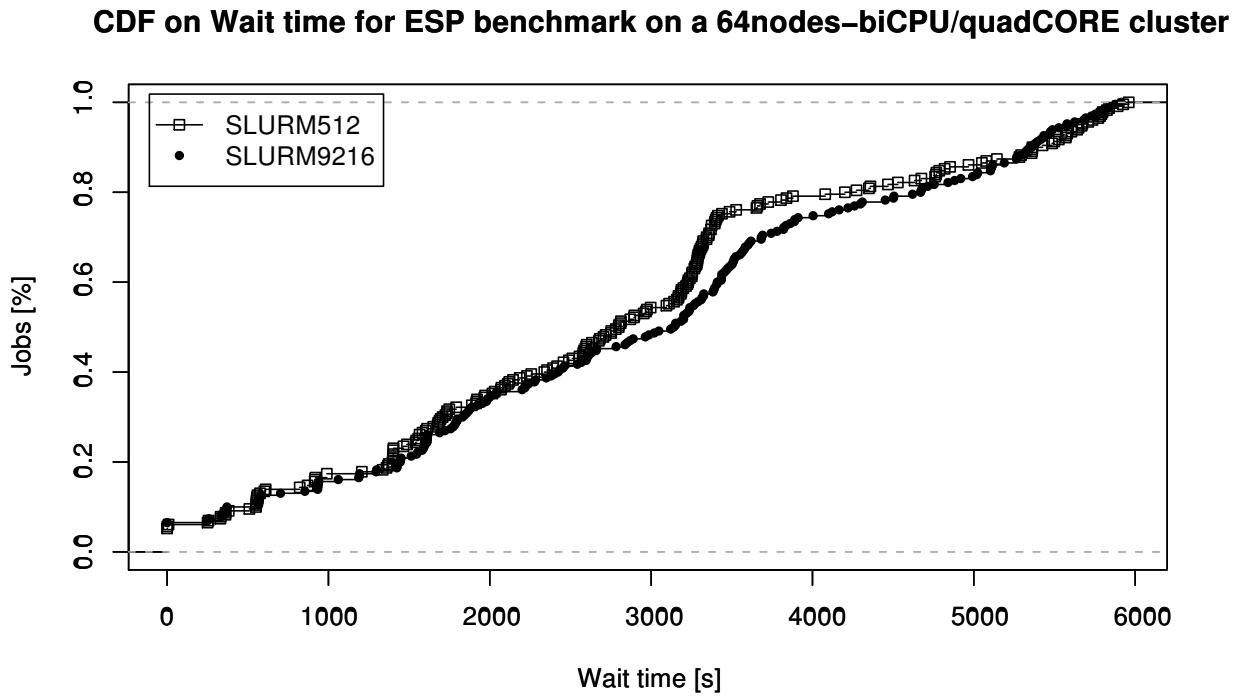


Figure 3.18: ESP benchmark CDF on wait time SLURM 512-9216 cores

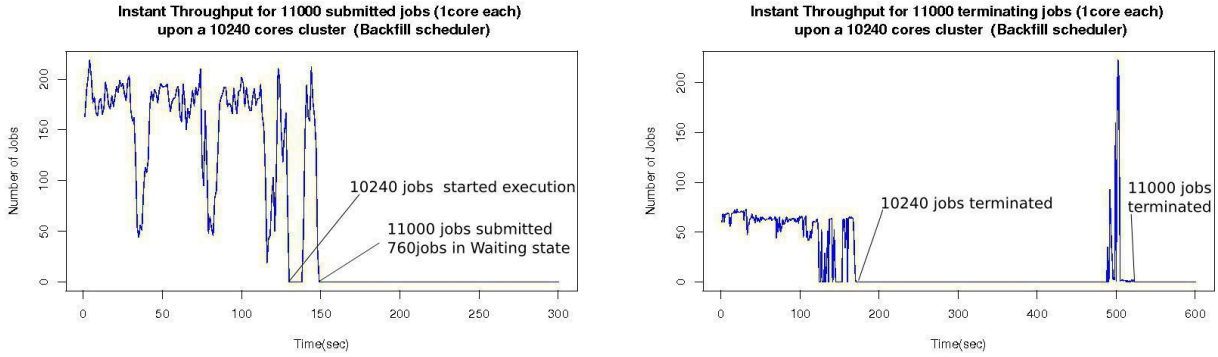


Figure 3.19: Throughput for submission(left) and termination(right) of jobs with SLURM - Backfill scheduler

different priorities between jobs so the preemption could not take effect) and we observed that the scheduler provided important problems and literally hanged after the submission of about 7500

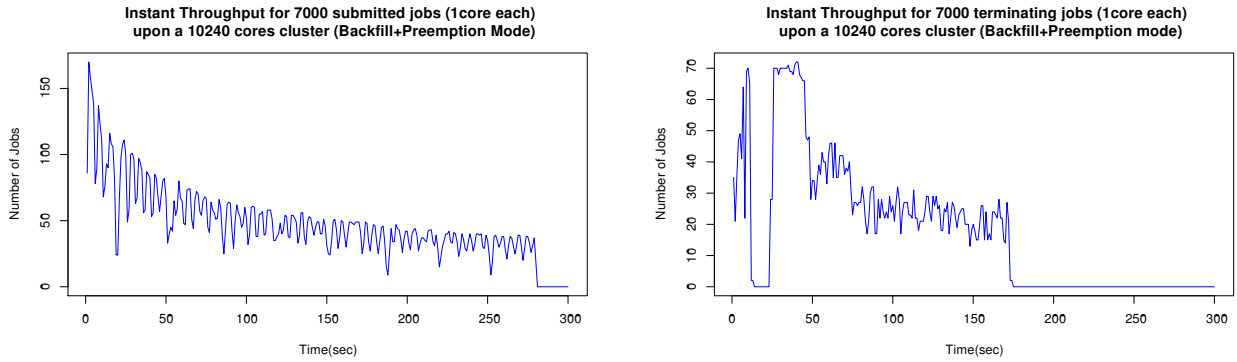


Figure 3.20: Throughput for submission(left) and termination(right) of jobs with SLURM - Backfill+Preemption scheduler

jobs. Hence a submission of 11000 jobs upon the same cluster with backfill+preemption policy was not possible and after several repetitions we obtained correct behaviour without system hanging at the limit of 7000 jobs. The measures obtained are shown on figures 3.20 The degradation of performance in throughout is obvious in both figures in comparison with the relevant for simple backfill scheduler. As might be expected, it is straightforward to understand that the preemption logic induces additional complexities on the scheduler and that's why how this degradation in throughput is explained. However we argue that since no higher priority job is used and no preemption is demanded throughout the whole experiment the scheduler should be able to perform its function with same complexity and results as the simple backfill scheduler. The complexity of scheduling jobs with preemption should be induced only when preempted and preemptable jobs exist.

Network Topology aware placement

For the evaluation of the efficiency of the Topology Plugin we have performed some changes on the default ESP benchmark by making use of NAS NPB-MPI parallel applications. As described on 2.3.2 we needed to particularly modify ESP benchmark to use applications with their normal target run times. Hence the new modified ESP benchmark, named *TOPO-ESP-NAS*, has been created with the following characteristics. The use of NAS application (CG class D) has been selected in place of the default `pchksum` MPI toy application, in order to be able to make use of applications that perform intense network communications. Moreover we deactivate the predifined execution time and we let the application perform the whole execution so as to measure its actual turnaround time. Our goal here is to observe the results upon application performance and overall

workload efficiency when making use of the topology aware best-fit plugin and to compare it with a similar execution without the use of the plugin.

The NAS parallel benchmarks [66], are widely used to evaluate the performance of parallel supercomputers as described in section 2.4.5. In our case we make use of only CG (Conjugate Gradient) benchmark which according to previous studies [67] is one of the most sensible ones to network contention.

Job Type	Fraction of Job Size relative to total system size	Job size for a 4096 cores cluster (in cores)	Least number of needed islands	Count of the number of total jobs	Percentage of job instance	Target Run Time (in sec) for execution upon 1 island	Target Run Time (in sec) for execution upon 2 islands
A	0.03125	128	1	75	32.6%	212	215
B	0.06250	256	1	9	3.9%	147	152
C	0.50000	2048	1	3	1.3%	374	523
D	0.25000	1024	1	3	1.3%	711	1143
E	0.50000	2048	1	3	1.3%	374	523
F	0.06250	256	1	9	3.9%	147	152
G	0.12500	512	1	6	2.6%	870	878
H	0.15820	648	1	6	2.6%	1003	1001
I	0.03125	128	1	24	10.4%	212	215
J	0.06250	256	1	24	10.4%	147	152
K	0.09570	392	1	15	6.5%	167	165
L	0.12500	512	1	36	15.6%	870	878
M	0.25000	1024	1	15	6.5%	711	1143
Z	1.00000	4096	2	2	0.9%	314	481
Total				230			

Table 3.10: Topo-ESP-NAS benchmark characteristics for 4096 resources cluster

SLURM NB cores-TOPO Cons / Topo-ESP-NAS-Results	Theoretic- Ideal values	4096 NO-Topology Aware	4096 Topology Aware
Total Execution time(sec)	12227	17518	16985
Average Wait time(sec)	-	4575	4617
Average Execution time(sec)	-	502	483
Efficieny for Topo-ESP-NAS	100%	69.8%	72.0%
Jobs on 1 island	228	165	183
Jobs on 2 islands	2	65	47

Table 3.11: TOPO-ESP-NAS benchmark results for 4096 resources cluster

The experiments have been effectuated upon the same cluster as our last experiments with a configuration of 128 computing nodes with a total of 4096 resources (cores) and 1 machine use as SLURM central controller. The network topology architecture consists of a fat tree with two islands of 64 nodes (2048 resources) each. The specific fat tree has a certain pruning level between the top switch and the medium level switches which means that there is not an all-to-all communication between the nodes of different islands.

The ESP benchmark table of job classes has been adapted for 4096 cores system and the run-times provided are the average execution times observed after 10 repetitions (with standard deviation less than 3/%) of jobs execution of CG benchmark class D upon the relevant number of resources. The values of these execution times are used to provide the best case for the *Total Execution* time (=12227sec in case all jobs are executed upon 1 island) of the new TOPO-ESP-NAS benchmark as provided on the second column of table 3.11. As we can observe the pruning effect can particularly influence the application performance on jobs with sizes more than 1024 resources as we can observe on the last two columns of 3.10. This is because of the particular topological interconnection which allows small jobs to have the posibility of an all-to-all communication even if they are placed to different islands.

The actual experimentation that were performed showed a marginal advance on *Average Execution* time along with a slight degredation in *Average Waiting* time when comparing the *Topology-Aware Placement* with the *No-Topology Aware* case. This is followed by an overall advance in Efficiency in Topo-ESP-NAS benchmark for the *Topology-Aware Placement* case. This difference is due to the fact that the Topology aware scheduling resulted into 79.6/% of the total jobs placed upon 1 island whereas in case of No-Topology aware this percentage is 71.7/%. Hence we have a difference of 7.9/% of jobs that benefit of a better placement for faster communication and possibly faster application performance.

Nevertheless the difference in *Topo-ESP-NAS overall Efficiency* is marginal. This is due to the fact that the pruning level does not really have an effect in all job sizes as we can observe on table 3.10 and the small number of islands makes performance differences quite similar for both cases. Ofcourse a tradeoff that needs to be considered is the Average Waiting time. The observed degredation is expected due to the imposed overhead to the scheduler to take into account topological characteristics in its algorithms. We need to observe this tradeoff between Execution time and Waiting time since it will play an important role in the overall efficiency results of the topology aware plcament techniques. However we expect that larger networks with larger depths and bigger number of islands will definately provide an additional overhead to the *Topology Aware Placement* plugin but also faster execution times for applications that are optimally placed and this will lead to differences on the final *TOPO-ESP-NAS* efficiency.

3.6 Conclusions

Management of Resources and Jobs for high performance computing in modern architectures has become a complex procedure with interesting research issues. Opensource and commercial Resource and Job Management Systems have been evolving to provide efficient exploitation of the infrastructures along with quality of services to the users. In this chapter we have made an in depth analysis of the internals of a RJMS and we have discussed about current research issues. A conceptual analysis and comparison of various opensource and commercial RJMS has been provided and some principal differences between the systems have been discussed.

In this chapter, additional light have been set to some research issues. The particularities of both inter-node and network topological architectures have to be taken into account to avoid application performance degredations. The overall system energy consumption needs to be considered and power-efficient management of resources need to be adopted. The increase of number of resources and jobs enforces the scalability issues of the launching and scheduling mechanisms of the RJMS. System High availability and techniques for jobs execution fault-tolerance need to provide certain guarantees especially in large-scale system were the possibilities for failures are more important. Various techniques are adopted by the different RJMS to deal with these issues. Particular experimental results upon specific RJMS gave us better insight for the understanding and analysis of these functionalities and provided performance evaluation comparisons bettween the RJMS.

Furthermore an analytical functionalities comparison between RJMS has been presented along with quantifiable evaluation scores which resulted to providing the best overall score for functionalities support to LSF commercial system. The second best score was reached by SLURM and the third and fourth position by OAR and SGE software. Closer look to the results showed that LSF provides the most evoluated Job Management System whereas SLURM has the better Job Management and Scheduling subsystems. However we argue that based on the constant evolution of technologies and software systems those results should not be taken as granted but more as an reference base for helping researchers to evaluate and compare the functionalities of Resource and Job Management Systems. This quantifiable evaluation can be utilized as the continuity of the research that has been started on 1994 by Kaplan [73], and Baker [74] and that it can be upgraded accordingly with the future technological evolutions and needs. Moreover the real-scale performance evaluation can be extended to be used for all different RJMS and provide complete performance evaluation results for all systems.

In our case, our principal interest is mainly upon the two systems that we have been studied extensively OAR and SLURM. The one aims more on simplicity and versatility. The second has as

goal scalability and efficiency. SLURM has been our testing system for scalability and efficiency upon large-scale infrastructures whereas OAR has been our testbed for the various prototype implementations that we propose on the following chapters. Both tools have their advantages and drawbacks and they are destined to different kind of environments. However we believe that both systems can take lessons from each other. The scalability issues of OAR scheduler observed during our evaluation process could be adressed in the future with the currently ongoing development of a new scheduler that benefits of a precompiled structural language. On the same time SLURM has been observed to be highly scalable in both number of nodes and jobs. However the slight throughput overheads of the backfill+preemption scheduler could be partially treated with the possibility of developping a scheduler per pending queue or partition (in SLURM terms).

Chapter 4

Improving system utilization in a lightweight grid context

Large scale scientific research, linking geographically distributed computational resources, is achieved by platforms known as *computational grids* and technologies termed *grid computing*. These infrastructures refer to a coordinated resource sharing of multi-institutional organizations with a common goal to perform particular scientific or technical large-scale computations [155]. They are often composed by multiple loosely coupled and geographically dispersed clusters with different administrative policies. Specialized software, termed as *grid middleware*, are used for the monitoring, discovery, and management of resources in order to promote the application execution upon the grid. At this level a collaboration between the local cluster resource and job management system and the grid middleware is needed to guarantee the local site autonomy.

The project with the greatest visibility on grid computing is Globus [156]. This mainstream approach, provides a software infrastructure that enables applications to handle distributed heterogeneous computing resources as a single virtual machine. It is based upon standardized services and formal allocation procedures established with the local RJMS of the cluster. Other approaches propose more *opportunistic* ways for the execution of large-scale distributed applications. Technologies like *desktop grid* or *volunteer computing* enabled scientists to perform computations more flexibly but with less guarantees, through the exploitation of idle cycles of desktop PCs connected on the internet. In a similar context alternative grid technologies proposed the idea of harvesting unutilised resources of multiple distinct administrative domains (clusters) that want to share their resources. This technology, usually termed as *global computing* was initially proposed by Condor [12] and implies the use of clusters' otherwise idle resources for execution of large-scale grid

applications.

In these environments, where no guarantee is provided for resources availabilities, the applications should be able to adapt themselves. A particular class of scientific applications known as bag-of-tasks is widely used under these contexts. They are composed by independent sequential tasks, able to use otherwise unutilized machines and would not be interrupted due to loss of resources. Furthermore their adaptability can play an important role on improving the systems utilization and reducing the external fragmentation.

Based on this context, this chapter presents CIGRI [25] which is a simple, lightweight, scalable and fault tolerant grid computing approach. It works discreetly on the interconnected clusters, without influencing the normal functionality of the local Resource and Job Management System. This is achieved by utilizing a transparent technique of harnessing the idle cluster resources for executing larger-scale computations. The platform supports all kind of parallel applications but our concern is centered on the use of bag-of-tasks applications.

One of the major challenges of the various approaches that exploit idle resources on grids, lies in their volatility. A common solution to deal with the high volatility of the resources on this kind of grid approaches, is checkpointing. In case of node failure or resource demand by a local user, the interrupted running task can be restarted on another resource from its last checkpoint. Hence, valuable computation does not get wasted and this may lead to faster execution of independent tasks and contribute to optimize the turnaround time of the whole grid application. Eventually, jobs smaller turnaround times may result into more efficient system utilization.

In this chapter, we explore the benefits of the checkpoint/restart technique as an optimization feature to the already existing fault-treatment mechanism of CIGRI system. Our contribution is based upon the BLCR [154] system-level implementation of checkpoint/restart. It consists of an application-transparent mechanism for rollback recovery on bag-of-tasks applications along with two different proposed strategies for the generation of checkpoints. Guided from a real-scale experimentation, with controlled parameters and trace files, we investigate the best strategy for optimization of the overall system utilization.

Based on the methodology described on chapter 2, we evaluate our prototype implementation of checkpoint/restart enhancements upon the lightweight grid CIGRI using real workload traces. Our experimentation is based on a large-scale deployment of the grid system under real-life conditions upon the French nationwide grid platform Grid5000 [157] using a real scientific Monte-Carlo type, bag-of-tasks application [70].

4.1 Background Information and Related Work

In this section we provide related work upon grid and alternative grid technologies, we discuss the research conducted in the areas of fault-treatment upon grids and scheduling for bag-of-tasks applications and we provide background information concerning the checkpoint/restart fault-tolerance technique.

4.1.1 Grid and alternative grid technologies

The worldwide research in Grid computing, has resulted in numerous different Grid packages and various approaches. Globus [156] toolkit, provides standardized services and capabilities to construct computational Grids. Most of research and commercialized projects on the grid that tend to evolve towards assurance and interoperablity with standards, are based on Globus toolkit. In this category we can find projects like Condor/G [158] which is a project that combines the security, resource discovery and resource access in grid environments as supported in Globus, with the computation management and harnessing of resources, on a single site, provided by Condor resource and job management system. In Condor/G, the use of cluster resources passes by a formal allocation procedure through the Grid Resource Allocation and Management (GRAM) protocol [159], which is part of Globus toolkit. However, the installation, configuration, customization and maintenance of a system like Globus, is a rather complicated task and requires a highly skilled support team, which not a lot laboratories are willing to afford.

The opportunity of performing large computations at low-cost, motivated scientists to come up with other technological solutions. The approach, usually reffered as *volunteer computing* or *desktop grid*, was based on the idea of harvesting the computing power (of individual desktop computers) going idle on the Internet. This technology, which is mostly used for bag of tasks applications, has become widely known by the application Seti@home [160]. The infrastructure that lies behind Seti@home is called BOINC [9] and was developed by the same team. This project aims to create a desktop grid computing infrastructure that can be used by several different desktop grid applications. On the same category we find WaveGrid [161] which introduced the idea of migrating jobs on available cycles of hosts located in idle night-time zones, around the globe. This work, addresses the fast-turnaround scheduling problem through migration strategies in peer-based cycle sharing system.

The work presented on [162] proposes a transparent resource allocation strategy to harness idle cluster resources in Computational Grids. This is part of a known project called OurGrid [11],

which provides an open, free-to-join, cooperative grid in which labs donate their idle computational resources in exchange for accessing other labs' idle resources. This project has to overcome security issues since it is based on the fact that there is no trust between the machine owners and the users that wish to use resources. In the same context, the "Flock of Condors" as presented on [12], provides a similar platform, with the difference that there is a significant amount of trust between grid users and cluster owners, permitting lighter security measures. In contrast to Ourgrid, this platform supports the execution of parallel applications.

4.1.2 Fault Treatment upon grid technologies

Computational grids provide an attractive platform for execution of complex computations. Nevertheless, due to their complicated characteristics (heterogeneity, complexity, distribution), they are more prone to failures than traditional computing platforms. There are many projects that try to address these technical challenges.

Medeiros et al. in [163] did a survey of failures in the grid environment and found out that for most of the grid users the greatest problem for recovering from a failure is to diagnose it. Phoenix [164], is a solution able to detect and possibly recover from failures in data intensive grid applications. It uses a "probabilistic" strategy to detect failures in file transfers and classify them into permanent and temporary. In the case of temporary errors, it tries to recover but in case of permanent it doesn't indicate actions to fix the system.

The work presented on [165] proposes failure analysis techniques which are used on the BlueGene/L IBM prototype. The focus lies upon the filtering and preprocessing techniques, used to substantially compress the error logs. In that way, it can accurately provide the failure occurrences of the system. In CIGRI case the fault-treatment mechanism approach, is influenced by this last project.

The most common mechanism to deal with failures on grids make use of rollback recovery techniques.

The project XtremWeb [10] is a desktop grid platform that provides fault-treatment mechanisms using the concept of concurrent RPCs and application-level checkpoints. The platform works effectively also in the case of parallel applications. Condor system [142, 166] provides fault tolerant mechanism through an advanced user-level checkpoint/restart library for Bag-of-Tasks and parallel applications. In this case the application needs to be compiled using the specific Condor libraries. HPC4U [167] is a Grid-enabled cluster middleware system. Its fault tolerance mechanism is based on the BLCR implementation of checkpointing and a virtualization toolkit, to provide

task migration on a cluster with different architectures. This is an enhancement comparing with our approach that does not provide task migration for different platforms. Furthermore, another difference is the fact that the checkpoint/restart mechanism is provided by the cluster resource manager, while in our case, it is provided on the grid level (CIGRI). Another interesting work regarding sharing checkpoints to enhance turnaround time on grids is presented in [168]. In contrast to our work, the former investigates only the case of application level checkpoints on institutional desktop grids and presents simulated results of their scheduling mechanisms. However, it is partly proprietary and in addition it lies upon mainstream grid approaches like Globus and the standards that are bound to it. Therefore it is rather different from our visions.

4.1.3 Scheduling for bag-of-tasks applications

The area of scheduling for bag-of-tasks applications upon volatile environments like *desktop grids* and *global computing* was subject of thorough research. Some systems are mostly interested for high throughput computing [12], while others provide mechanisms for optimization of jobs turnaround times [11], [169].

Kondo et al. in [170] analyzed several strategies for various configurations of institutional desktop grid environments. Strategies such as resource prioritization, resource exclusion and task duplication were compared and good results were observed for resource exclusion with makespan prediction along with task duplication. However the work considered only small-sized jobs which is not always the case and did not take into account checkpointing, assuming that interrupted tasks would be restarted from scratch.

Another project that proposes a similar method of task-duplication is OurGrid [11]. The proposed scheduling policy of OurGrid is named workqueue-with-replication (WQR). This policy, implies that tasks are assigned in a FIFO manner, regardless of the metrics related with performance of machines. When all tasks have been distributed to resources, and if there are enough free resources, the system creates replicas from randomly chosen tasks. This is different than the task duplication in the previous example where the duplication starts when the number of the idle resources is greater than the number of tasks to schedule. WQR does not guarantee the execution of all tasks but in comparison with simple FIFO policy it significantly augments the probability of an application being terminated.

Anglano et al. [169] propose a fault-tolerant version of workqueue WQR-FT. This policy extends the basic WQR methodology with checkpointing mechanisms. They recommend checkpointing usage for environments with unknown availability, since it yields significant improvements

when volatility of resources is high. Likewise WQR, WQR-FT is also limited by its best-effort approach, with no guarantee that all tasks comprising a given application are effectively executed.

The work in [168] exploits the use of private and shared checkpoints for the scheduling of bag-of-tasks applications upon volatile environments. Simulation results were obtained through the experimentation of real trace files. The results presented variations depending the time of the day but in general the shared checkpoints lead to faster turnaround times than the private ones. Furthermore strategies combining task replication and checkpoints were highly efficient especially on heterogeneous machines.

Weng et al. [171] study the scheduling of bag-of-tasks depending on the needed input data of each application. They propose the Qsufferage algorithm which considers the influence of input data repositories location on scheduling. The study confirms that the size of the input data of tasks has an impact in the performance of the heuristic-based algorithms.

One of the latest studies proposes [8] strategies for dealing with multiple bag-of-tasks applications which is an area that not a lot of research has been performed yet.

The scheduling method proposed in CIGRI is based upon a simple FIFO policy for the different tasks of the BoT application. Our implementation extends this polciy with tasks prioritization mechanisms for the tasks that provide checkpoints.

4.1.4 Checkpoint-Restart Recovery technique

A widely used solution to cope with the high failure rate of the computing nodes is the technique of checkpoint/restart. There are two main types of checkpoint: application level and system level. The first ones are recognized as more time and space efficient, while the use of the system-level checkpoints can offer benefits like application transparency and preemption.

BLCR is a robust open source implementation of rollback recovery capable to checkpoint a wide range of applications, without requiring changes to be made to the application code. It can be used either as a stand-alone system for checkpointing applications (multi-process or multi-threaded) on a single node or interfaced by a scheduling system (SLURM, Torque, etc), grid middlewares [167] and parallel communication libraries for checkpointing and restarting parallel jobs running on multiple nodes (MVAPICH2, LAM-MPI [172], OpenMPI). BLCR is implemented as a Linux kernel module which means that it performs checkpointing and restarting inside the linux kernel. One of its limitations is that it does not support open sockets (TCP/IP, Unix domain, etc.). The support of parallel and distributed applications is implemented through a mechanism to register user-level callback functions that are triggered whenever a checkpoint occurs, and that continue

when the process restarts. These callbacks allow the application to shutdown its network activity before a checkpoint is taken, and restore them later.

Since the time this study took place, the latest BLCR version [173] has provided significant advances in particular on the support of different kernel versions, architectures and features like unlinked open files, pid virtualization and other operating system artifacts.

4.2 An alternative lightweight grid computing approach for bag-of-tasks applications

CIGRI system provides an alternative, lightweight approach of a classic grid platform by aggregating the clusters' idle computing resources.

Back in 2002 when the project CIMENT CIGRI began, scientists of different disciplines (environment, chemistry, physics, astrophysics, biology, mathematics,etc) wished to *mutualise* the computing power of their private laboratories cluster resources, so that they could effectuate larger scale computations. However, the prominent mainstream solution of Globus seemed a very complicated and expensive solution for their demands. On the same time, the emergence of desktop grid systems and the idea of "cycle stealing" technologies, provided a good base for building our approach. CIGRI project was born in this environment. The result, has been a simple lightweight system based on a modular architecture and high level components that facilitates the aggregation of the clusters' locally unutilized resources, for execution of bag of tasks applications.

CIGRI software has been in production and an active open-source research project since 2002 [25]. In one of the cases where it is utilized, its users can benefit of the power of 6 different clusters and a total of more than 700 processors of heterogeneous machines.

4.2.1 Lightweight grid and cluster utilization policy

CIGRI software is based on the concept of lightweight grid, which are infrastructures that simplify the general problem of the grids. This simplification generally goes through a certain homogeneity of services and administration procedures: by adopting the same services and configurations on all interconnected clusters.

The main concern behind CIGRI system, is to provide a platform that focuses on the research and development of specific problems that come along with the execution of large-scale grid jobs (scheduling, fault tolerance,...) upon distributed grid environments. The initial environment for

which the system was designed, had been laboratories clusters, where users enjoyed a significant trust. Under those terms, our approach does not deal with critical problems inherent on computational grids like security, authentication mechanisms and resource location.

Another objective had been not to influence the normal functionality of the interconnected clusters by the use of CIGRI platform. In contrast to Globus and OurGrid approaches, no specific CIGRI software has to be installed on the clusters apart a local RJMS with some specific features. CIGRI software uses the Resource and Job Management System (RJMS) of each cluster and acts like a normal user to submit jobs upon the clusters and collect the results.

An important issue to treat is the policy concerning the use of the different clusters. Thanks to this grid approach, outside users can use the computing power of a cluster. However, the jobs submitted by CIGRI on the clusters, must not disturb the use of them by their local users. The idea is to use only the idle resources of the cluster. Thus the proposed solution was to introduce the concept of *best effort* tasks on the RJMS. More specifically, the jobs submitted by CIGRI on the different clusters are submitted with the help of the local RJMS as a special type of jobs named *best effort*. The jobs of this type have the minimum `null` execution priority and are submitted only if there is an available unutilized resource. Moreover, if during the execution of this job the resource is requested by a local cluster user, the CIGRI *best-effort* job is killed by the local RJMS and CIGRI grid server is notified. As we will see on next section, the fault-treatment mechanism takes the appropriate measures to resubmit the killed jobs and thus guarantees a successful completion of the whole calculation.

One other issue that emerged from the need of simplicity had been the type of applications that are supported. Initially CIGRI platform supported only bag-of-tasks applications but latest version have been enhanced with the support of all parallel applications. In this study we deal only with the case of bag-of-tasks applications through CIGRI.

4.2.2 Global Architecture

CIGRI software is composed by a server which communicates with all the Resource and Job Management Systems of the clusters. It works like a normal user, submitting minimum priority jobs as best-effort tasks. There is no specific CIGRI software installed on the clusters, but it makes use of the local RJMS for submission of jobs, execution control and monitoring of resources. Furthermore, it just takes advantage of the classic system tools (ssh, bash, sudo, cat, ls, scp, tar...) and the NFS file system of the cluster. Currently CIGRI supports only the OAR resource and job management system which has been also developed by the same team. However the interaction

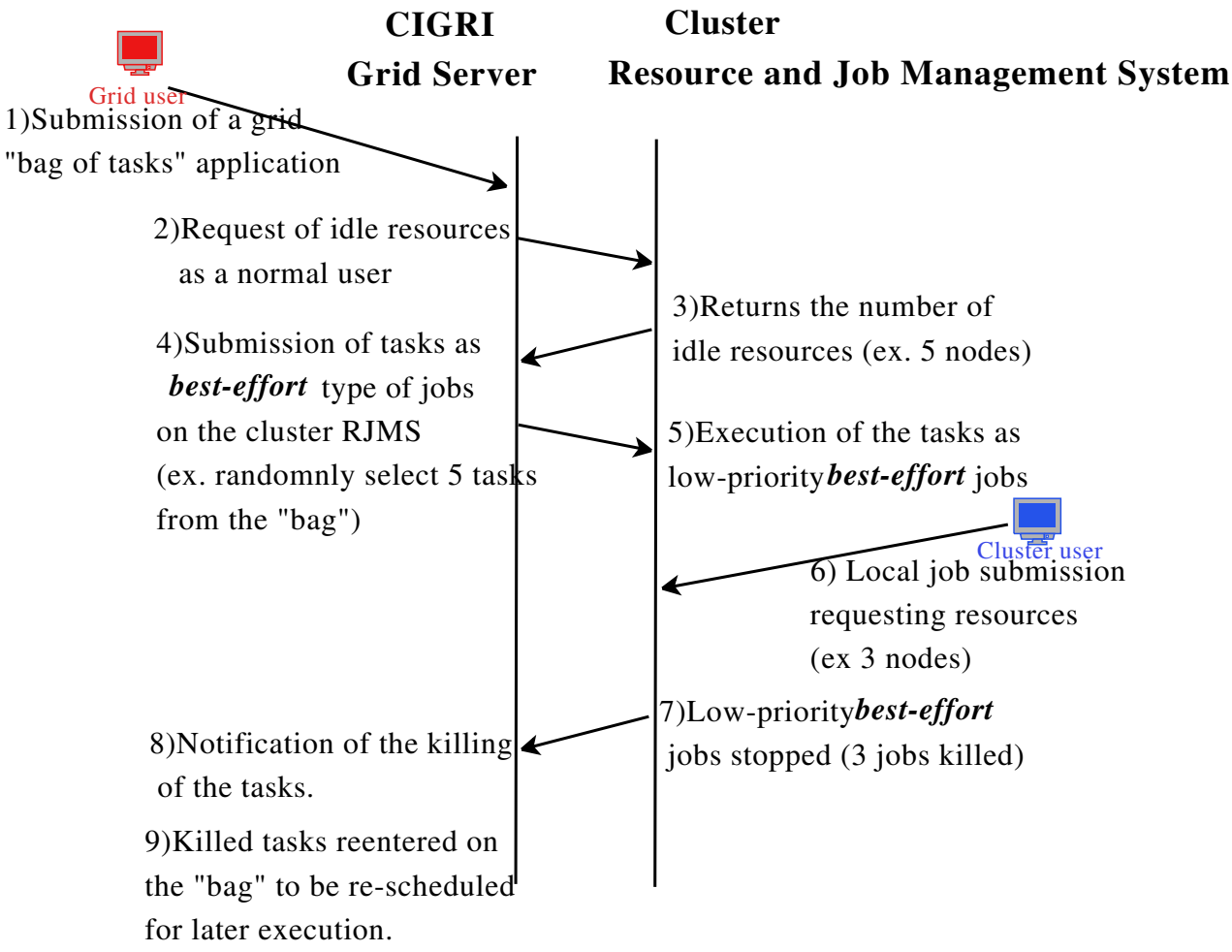


Figure 4.1: CIGRI cluster utilization policy: best effort jobs.

with other RJMS is possible if a feature like *best-effort* jobs exists on the RJMS. The *best-effort* type of jobs has been introduced to OAR to support the aggregation of idle cluster resources and the lightweight grid approach of CIGRI. The platform architecture is shown on figure 4.2.

High level componennts

The system is designed using a modular architecture based upon two high level components. Those are the SQL database and the scripting language Perl. The recipe of high level components, has already been tested and validated on the case of OAR in [174]. Similarly with OAR in CIGRI platform, the database is used to hold all the internal data and works as the only communication medium between the modules.

It is used to store information such as: the state of clusters' nodes, the state of submitted jobs,

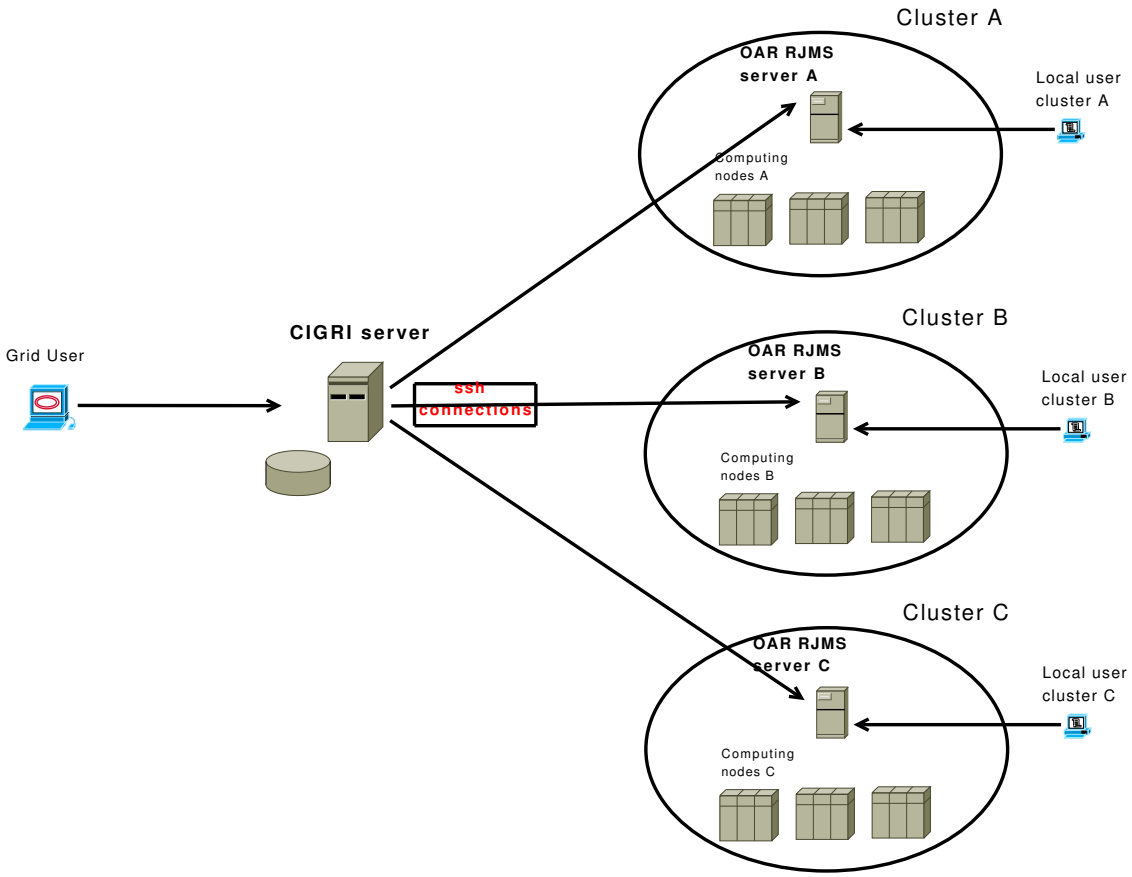


Figure 4.2: CIGRI platform architecture

the various errors that can be obtained and logged, logging of all activities, information about the users, etc. The choice of a database as the central component, ensures the management of big amounts of information and guarantees the reliability of the platform.

Modular Architecture and functionalities

The central part of CIGRI is made of a collection of independent modules. Each of them is in charge of a specific task. Tasks such as jobs scheduling, jobs monitoring, results collecting, error logging are all handled by separate modules. They all interact with the system using the database. The whole system is managed by a central module which is in charge of calling the other modules to perform either regular tasks (such as monitoring) or on-demand tasks (such as submission). Figure 4.3 shows the design of the several modules and describes their functionalities

For the transfer of the necceasary data of the application from the grid server to the clusters, the specific *Cluster Synchronizing* module is responsible for automatic and transparent transfer of

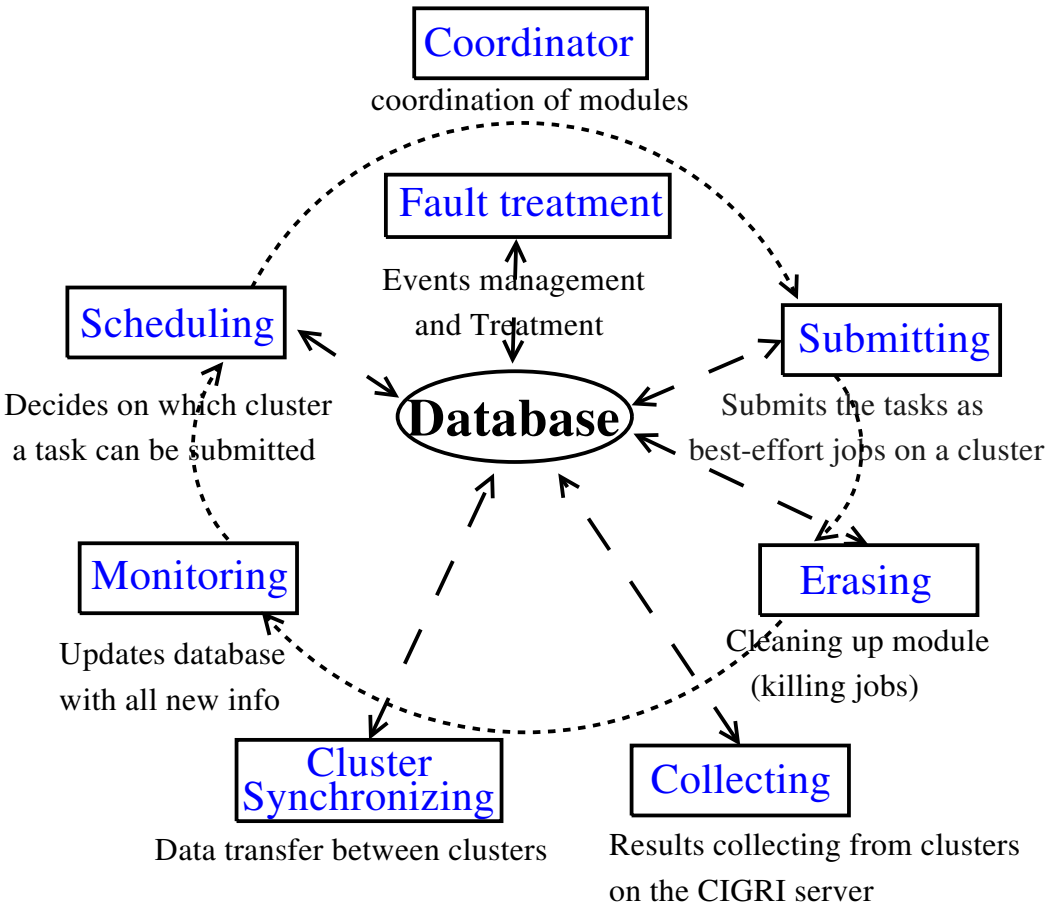


Figure 4.3: CIGRI internal modules and their operation.

data to all the clusters that will take part on the computation. This module in cooperation with the *Scheduling* module guarantees that tasks will start their execution on a specific cluster only if the necessary data are completely transfered.

Another important functionality is the results' collection from the clusters, along with the cleaning of temporary files. A specially designed module, auxiliary to the system is responsible to periodically collect the result files and archive them on the grid server and on the same time erase them from the cluster. Hence, the user can collect the results of its computation on one centralized

place and on the same time the cluster's storage resources are not overbooked by grid application data. Moreover, a web portal exists to monitor the jobs on the grid.

Fault treatment mechanisms

In Cigri fault treatment mechanism the initial concern is to be able to locate, log and categorize the different errors that are possible to occur. A possible error is located by the mechanism and logged as an event type in the database, for further analysis. To facilitate the treatment of these events we decided to classify them into 4 different classes (depending on the treatment that they should have). Thus, we have errors that are relative to the Scheduler, the Clusters, the individual submitted tasks and finally to the grid bag of tasks jobs. In that way, every different module controls its own possible anomalies or errors by providing control checks and execution timeouts to its different commands. To ensure the platforms' reliability, the Fault Treatment mechanism provides automatic actions to be taken for every different category.

Taking as example an error that occurs due to a problem on the cluster RJMS; Since Cigri system does not interfere directly with the clusters' functionality, the decision to be made is to automatically stop the usage of this cluster, by *blacklisting* it, on the grid level and inform administrators, with automatic email, to restore the cluster into its normal operation. While the usage of a cluster is restricted, the execution of the applications can be continued on the rest of the clusters. The mechanism works similarly for any errors may occur in the "Scheduler" or "Cluster" classes.

However, there are types of events that can be automatically and transparently treated. This happens, for example, when a grid best-effort task is killed by the local cluster RJMS, so that the resource can be occupied, by a higher priority local cluster job. In this case, the Monitoring module detects that the job was killed and issues a specific event which is logged and treated by the Fault-Treatment module. Hence, the task is reentered on the bag of tasks to be submitted when and where it can be rescheduled (default strategy). This happens when there are free resources on any of the clusters and the Scheduler chooses, in random, the specific task.

4.3 Enhancements for turnaround time optimization

Grid approaches like volunteer computing, Condor, OurGrid and CIGRI suffer much more failures than those met on mainstream grid technologies. Apart from the volatility failures of the resources (network outages and machine crashes), the most common interruptions of job executions are provoked by *interference* failures. Those failures occur when the local user claims utilization of resources that are used opportunistically for a grid job and as a consequence, the particular tasks

have to be interrupted. Under specific circumstances, this can lead to very large turnaround times for particular bag-of-tasks applications that their tasks are continuously interrupted. In this case, hours of computation can turn out *wasted* and the utilization of otherwise idle resources will not be that beneficial after all.

In order to deal with these failures and provide a *beneficial* utilization of the otherwise idle resources, we propose a turnaround time enhancement upon the lightweight grid solution CIGRI, through the use of a checkpoint/restart mechanism.

4.3.1 Transparent Checkpoint/Restart mechanism for a lightweight grid

The enhancements that we have designed and implemented for the CIGRI lightweight grid platform are based upon the widely used system-level implementation of checkpoint/restart BLCR [154, 154]. We have taken advantage of its transparent capabilities of supporting checkpoint/restart upon a wide range of applications. BLCR provides a user-level library along with specific commands for checkpointing (`cr_checkpoint`) and restarting (`cr_restart`) applications. In order to make the application checkpointable through BLCR, a particular command (`cr_run`) has to be used, that loads the BLCR library into the application at startup time.

The Cigri scheduler module collects the idle resources from each cluster and designates the tasks that are going to be executed upon each cluster. Once this decision is made another module (Runner) is responsible for preparing the task as an OAR `best-effort` job and send it to the cluster. Our design to provide Checkpoint/Restart through BLCR is composed by a wrapper that packs up each task executable with the particular BLCR commands. In particular the job is submitted as a simple bash script that makes use of the `cr_run` command and a signal trapper that can trigger a checkpoint generation through `cr_checkpoint`. Once the application has begun execution a transmission of a linux signal (SIGUSR1) upon it will be trapped and the `cr_checkpoint` command will be triggered which makes the application to checkpoint itself. The checkpoint file produced is initially stored on a temporary directory and then moved on the NFS file system of the cluster. In this case, NFS facilitates a possible job migration on another homogeneous node of the same cluster. As we will see on the following section, our design provides different methods for the timing of the checkpoint generation and this is related with the moment that this signal is transmitted.

The default fault treatment mechanism of CIGRI system, as mentioned on 4.2.2, implies that the interrupted task will be reentered into the 'bag' (of tasks) and will wait to be rescheduled whenever there is an idle cluster resource available and it's turn for execution arrives. This policy

has been slightly changed for the checkpoint/restart scenarios, in order to favor the jobs that have generated checkpoints. The difference is that a checkpointed job will begin execution from it's last valid saved state thus gaining important computation time. Indeed the checkpointed jobs are closer to termination and that's why they are treated in priority by the scheduler. The particular task is prepared using a similar bash script using the BLCR command `cr_restart` The use of both `cr_run` and `cr_restart` in the same script allows the application to be restarted from a valid checkpoint and still remain checkpointable through the trapping mechanism.

4.3.2 Periodic and Triggered checkpoints strategies

Our design provided two different strategies for the checkpoint/restart mechanisms support. The strategies are defined by the specific timings that the checkpoints will take place. Figure 4.4 provides a view of the default fault-treatment strategy, while figures 4.5 show a graphical representation of the different enhanced checkpoint/restart strategies.

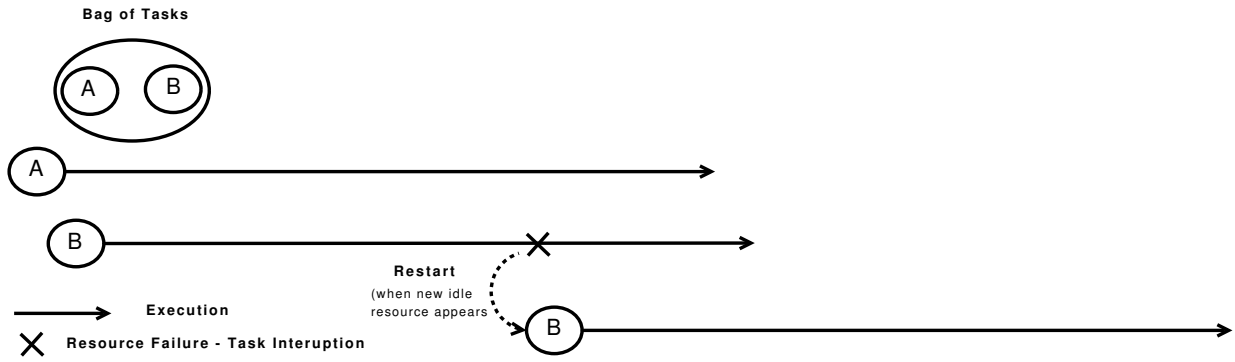


Figure 4.4: Fault treatment default strategy

Periodic Checkpoints

The first strategy is based upon a mechanism that allows the periodic checkpointing of the different tasks. This mechanism is introduced by the inclusion of a parameter on the submission of the bag-of-tasks grid job which will determine the periodicity (in seconds or minutes) that the checkpointing should take place. Thus the job executed on the cluster, powered by the special BLCR commands, can checkpoint itself and the periodicity of the checkpoint procedure depends on the value provided by the user. The advantage of this strategy is that it permits the application to

be fault-tolerant for every kind of failures, either interference or hardware ones. On the other hand system-level checkpoint implementations have the drawback of the big overhead. The checkpoint file can be large and the duration to checkpoint can also take a lot of time. Therefore, taking a big number of checkpoints on jobs that finally are not killed, can be a big waste of time, crucial for the final turnaround time of the whole computation.

Triggered Checkpoints

In order to treat the drawbacks of the Periodic Checkpoints strategy we have designed a second strategy more flexible but more constrainted than the previous one. This one provides fault-tolerance by placing more emphasis on the final turnaround time of the computation. The *Triggered Checkpoints* strategy addresses only the interference failures, which under normal conditions on this context, are the most common. In this kind of failures the local RJMS kills immidiately the besteffort job and gives the resources to the higher priority job. Our proposal is to add to the RJMS the capability of notifying the specific "to be killed" job, so that it can checkpoint itself, just before it gets interrupted. Following this approach we ensure that every checkpoint taken, will worth the overhead.

This *grace* time delay that the RJMS should wait for the checkpoint procedure, before it removes the resources from the job has been an important issue of this method. At the time of this study other studies [175] were underway to investigate the relevance among the memory that a particular task occupies during its execution and the size of the produced checkpoint file along with the duration of the checkpointing procedure. Considering these values we could deduce the time delay that the RJMS worths waiting before it "kills" a job. In our case, without the results of those studies, the choice was taken empirically and set to a specific small constant value (4sec) according to the particular application that we used for experimentation. Ofcourse we argue that a *grace* time delay beyond 20sec can no more be considered as *grace* time by the RJMS so the support of really complex and large applications is not provided. So another drawback of this method is that it supports rather simple applications that do not fill up the memory while executing and do not produce large checkpoint files. On the other hand, applications that fit the supported characteristics could benefit from guaranteed small turnaround times.

4.4 Experimentation Results

Based on the experimental methodology that we have developed, described on chapter 2, we have effectuated real-life large-scale evaluation of our prototype. We have deployed CIGRI as

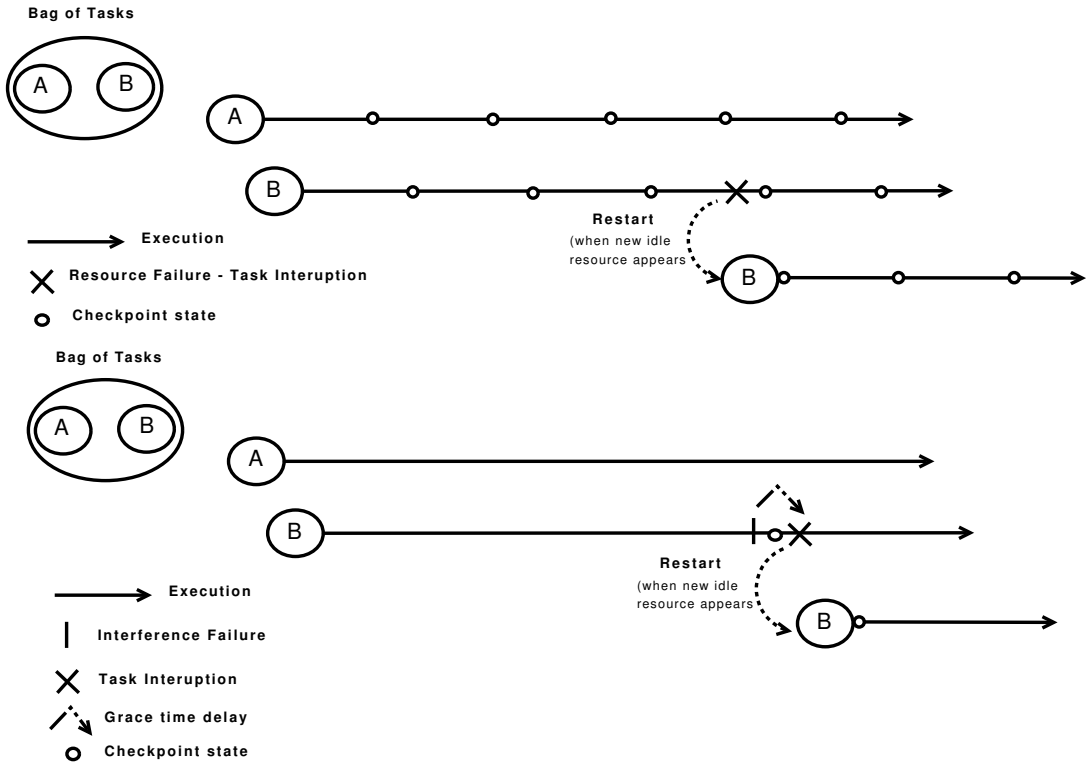


Figure 4.5: Fault treatment strategies: Periodic checkpoints (top), Triggered checkpoints (bottom)

a lightweight grid of 5 clusters. The experiments were based on the one hand upon real production workload traces of a particular grid platform DAS2 [5] representing the local jobs of each cluster; On the other hand upon a real astrophysical bag-of-tasks application MCFOST [70] submitted to CIGRI for large-scale computation, representing a grid job. Depending on the different strategies used we observe the valuable and wasted computation during the exploitation of idle resources by the CIGRI approach.

4.4.1 Deploying a lightweight grid upon Grid5000 platform

Taking advantage of the capability of environment deployment based on kadeploy toolkit of Grid5000, as described on chapter 2 a user can very easily deploy his own cluster or even grid upon the Grid5000 platform. In more detail, for our experiments we constructed an environment containing all the needed software (OAR, CIGRI, NFS, BLCR, ssh,...) installed and ready for configuration. Once the environment is deployed on all the allocated nodes, we proceed to the software configuration and the attribution of the roles (Grid server, Cluster server, computing node, ...) just by

choosing the specific services to be launched to each deployed node. Hence, finally we obtain a grid infrastructure ready for experimentation.

Grid5000 is composed of 11 sites from 1 to 4 clusters each one of them and various architectures. Since the choice of our experiments infrastructure has to be identical to this of the DAS2 DUAL CPU nodes grid platform, we have chosen a site with DUAL CPU (one core) Opteron architecture on 2.0GHz or 2.4GHz and 2MB of memory. In more detail DAS2 grid platform [5] is consisted of 4 clusters of 32 computing nodes each and 1 cluster of 72 nodes. In our experiments we deploy a grid, identical to DAS2 grid platform, using the nodes of one Grid5000 cluster (the GDX cluster at Orsay). For the needs of the experiment we deploy: 132(= 32 x 4 + 4) similar nodes (Opteron 2.0GHz) to represent the 4 clusters (of 32nodes) along with their 4 cluster RJMS servers; and 74(= 72 + 1 + 1) similar nodes (Opteron 2.4GHz) to represent the 1 cluster (of 72 nodes), its RJMS server and finally one more node to be used as the CIGRI grid Server.

Our initial concern was how we could choose specific parts of the workload traces that we could use on our experiments. As described on chapter 2 (section 2.4) the criterion for that, was decided to be the percentage of the used resources in relevance with the total amount of them throughout the grid, for a specific duration. Hence, according to our desired duration of the experiment, we can decide high cluster user load (80%) or low cluster user load (15%). This choice allows us to experiment the function of CIGRI under various conditions. The jobs submitted by Xionee (chapter 2-section 2.5) toolkit, have the corresponding characteristics (allocated procs, launch time, end time , duration, ...) of the different workload traces of the DAS2 grid platform. However for the sake of these experiments no particular task was sent for execution for these slots of time. Hence, the local cluster jobs perform no real computation, they just occupy the resources for a specific duration (sleep jobs). However, these local workloads provide the *interference* failures (towards the grid bag-of-tasks job) that are needed for the observation of the function of the lightweight grid experimentation testbed.

In contrast with the local cluster jobs which don't make real calculations, the grid submitted application performs valuable computations. In more detail, the bag-of-tasks application submitted to the grid through CIGRI is a benchmark of a 3D radiative transfer code based on Monte-Carlo method, called MCFOST [70] (described on chapter 2 section 2.4).

Concerning the results presented on the following section we submitted 30000 tasks of 45min of computation each (on one Opteron CPU 2.0Gz, 2MB of memory) on every different experimentation. Since our testbed lasts only 5 hours, the completion of the computation was impossible but the goal was to successfully execute as many tasks as possible.

4.4.2 Performance Evaluation

The purpose of the experiments presented has been the testing of the new checkpoint/restart feature as a fault-tolerance mechanism and an optimization method of the overall turnaround time of the computation of a bag of task application. In more detail, our experiments are driven by a big number of parameters and conditions that can be selected before the start of the experiments. The *grace* time delay of the RJMS, discussed on Section 4.3, for the *Triggered Checkpoints* strategy was set to 4sec and the checkpoint periodicity for the *Periodic Checkpoints* strategy was defined to be 20min. These values were defined empirically after observations and were selected as optimized cases for each strategy.

Since the cost of real-life experimentation is high, we initially constructed small scale versions of the experiments, so that we can better understand the function of the checkpoint feature and observe the differences between the strategies. Hence, we experimented with a grid that contains only one 32nodes cluster and was driven by a 5hours part of a DAS2 trace. Figure 4.6 presents 3 experiments each representing a different strategy *Triggered Checkpoints*, *Periodic Checkpoints* and *No Checkpoints* (default) for grid fault-tolerance. The experiments were driven by the same trace representing 40% local cluster utilization.

The y-axis represents the jobs execution time multiplied by the occupied resources. For example on a 64CPU cluster, occupied for 5hours, the maximum usage can be 1152000 CPUxsec. The line is the difference between the maximum cluster utilisation potential minus the local cluster usage, thus represents the maximum utilization that can be effectuated by the grid users. The first bar on each strategy named as `Total` represents the total system utilization of the grid besteffort jobs on the cluster. The big gap between the edge of the bar and the maximum cluster utilization potential (red line), on the second experiment, shows a very low efficiency in exploiting the cluster idle resources. The reason is the big checkpoint overhead along with the big number of useless periodic checkpoints.

The bars `Terminated`, `Running` and `Error` represent the system utilization of jobs that were found on these respective states after the end of the experiments. The bars `Fail.NotValuable` and `Fail.Valuable` represent the cluster utilisation of jobs that were failed due to interference by a higher-priority local cluster job. The cluster utilization is *valuable* if the job is restarted from a checkpoint and finally arrives to termination. In case of a checkpoint error or inexistance or if simply we use the default strategy without checkpoints, this cluster utilization is wasted and is represented by the `Fail.NotValuable` bar. Indeed, on the default fault-treatment mechanism *No checkpoints* once the job is killed the already produced work turns out to be completely wasted.

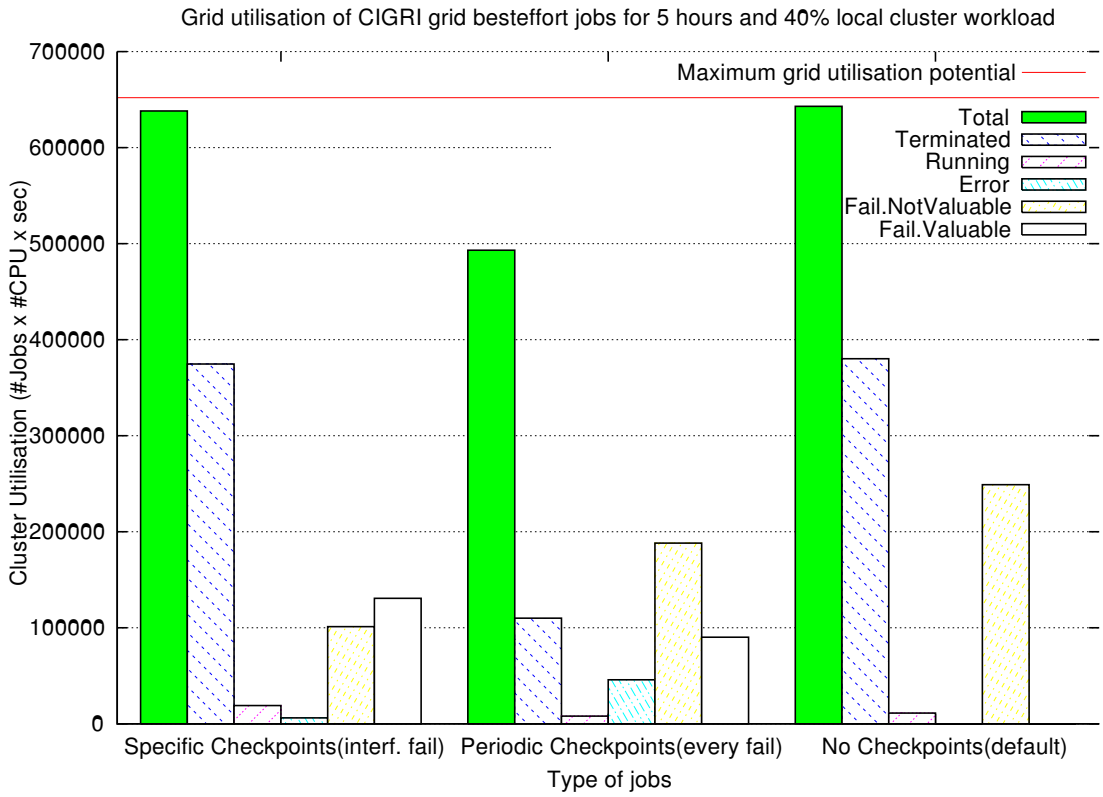


Figure 4.6: CIGRI grid utilization for 1 cluster of 32nodes grid (3 strategies comparison)

Strategies / Jobs	Total	Terminated	Running	Error	Inter. Failures Not Valuable	Inter. Failures Valuable
Triggered checkpoints	338	**156**	14	20	70	78
Periodic checkpoints	302	50	6	45	168	33
No checkpoints	343	**134**	14	1	194	0

Table 4.1: State of grid jobs for 5hours experiments of 1cluster, 32nodes, 40% local workload and 3 strategies

The sum of the system utilizations represented by `Terminated`, `Running`, `Error`, `Fail.NotValuab` and `Fail.Valuable` for each strategy results in the total system utilization of each one of them. It is interesting to observe on figure 4.6 how a part of the *NotValuable* utilization of *No Checkpoints* strategy becomes *Valuable* utilization on the *Triggered Checkpoints* strategy. Respectively

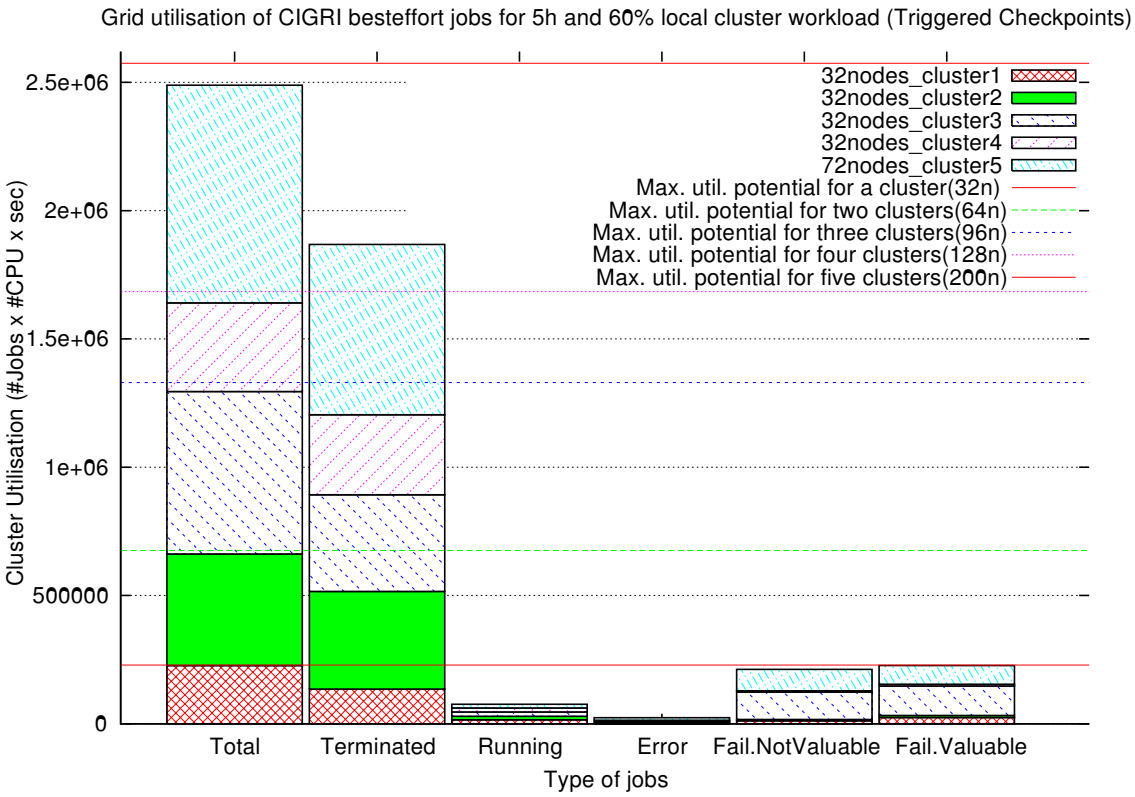


Figure 4.7: CIGRI grid utilization for 5 clusters of of 200nodes grid, Triggered checkpoints strategy

we can see how the *Periodic Checkpoints* method is limited by its overheads and how this influences the overall usefull exploitation of the cluster. Table 4.1 shows the final results with the `Total` number of treated tasks (of the bag-of-tasks application) figuring on the first column and the number of tasks for each relevant state is represented on the following 5 columns The last 2 columns `Inter. Failures Not Valuable` and `Inter Failures Valuable` represent the tasks that have been interrupted and rescheduled with wasted or valuable computations respectively. It is interesting to see how the Triggerred checkpoints strategy results into bigger number of Terminated jobs and thus to a more efficient exploitation of otherwise idle resources.

The experiments were then reapeted on the real scale of DAS2 grid platform. Under the same context, figures 4.7 and 4.8 present a large-scale experimentation of the CIGRI grid platform exploiting the resources of 5 clusters which are driven by the DAS2 grid platform traces. The two graphs represent a different fault-tolerant strategy for the same parameters and conditions (5hours of experimentation, 60% local cluster grid workload). In this second phase of experimentations we

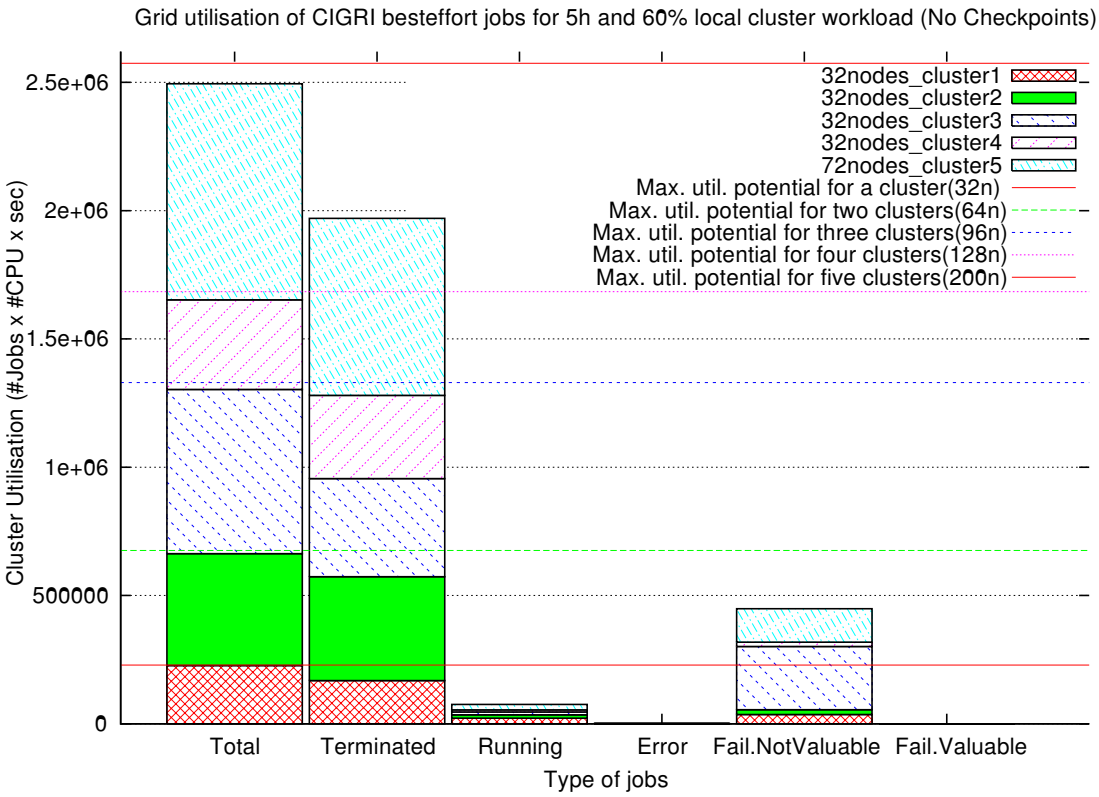


Figure 4.8: CIGRI grid utilization for 5 clusters of of 200nodes grid, No checkpoints strategy

have experimented only with the triggered checkpoints and the default strategy, since the periodic checkpoints gave us poor results on the first phase.

Like on the previous representation of results, the different lines represent the maximum cluster utilisation potential that can be effectuated by the grid users according to each clusters local workload. It is interesting to see that the grid utilization of the Terminated jobs is less on the *Triggered checkpoints* than the *No checkpoints* strategy. This is due to the fact that, in the case of the checkpointing strategy, a job restarted from a checkpoint takes less time to terminate its execution than a normal uncheckpointed job. As a matter of fact the grid utilization of those checkpointed jobs, that were restarted and 'Terminated' successfully, is represented by the `Fail. Valuable` bar. As we can see from the table 4.2, we observed 212 checkpointed jobs ("Inter.Failures Valuable") that their checkpoint was used for a successful termination of a job in contrast with the 581 jobs ("Inter.Failures Not Valuable") of wasted computation of the *No checkpoints* strategy. The same

Strategies / Jobs	Total	Terminated	Running	Error	Inter. Failures Not Valuable	Inter. Failures Valuable
Triggered checkpoints	1377	**762**	74	103	226	212
No checkpoints	1420	**739**	74	8	581	0

Table 4.2: State of grid jobs for 5h experiments of 5clusters, 200nodes, 60% local workload and 2 strategies

gain can be observed on the relevant bars of figures 4.7 and 4.8. Hence, the result is a larger number of Terminated jobs for the *Triggered checkpoints* strategy, for the same experimentation time duration (5hours).

Obviously, the gain regarding our expectations, is marginal 761 terminated jobs with the Triggered checkpoints strategy versus 739 with the No Checkpoints strategy. Nevertheless, the explanation lies partly on the really large number of 'Error' state jobs observed on all our experiments with checkpointing strategies. Indeed, the tasks of the specific astrophysical application sometimes could result into Segmentation Fault, but the probability of this Segmentation Fault becomes higher if the job is checkpointed and then restarted using BLCR. This bug is currently under investigation. Finally, for our future experimentations we are considering not only various scenarios but different applications as well. We effectuated repeated experiments to observe the behaviour of the different fault-treatment strategies by changing the local cluster workload percentage and the duration of the executed tasks of the application submitted to the grid. The experiments shown in this chapter represent a good sample of our total observations.

4.5 Conclusions

In this chapter we studied the effects of a system-level checkpoint/restart mechanism as enhancement of the scheduling and fault-treatment policies of a lightweight grid platform for bag-of-tasks applications. Driven by the motivation of improvement of the exploitation of the cluster, we proposed different strategies to guarantee that the optimization of system utilization is also beneficial for the users and contains less wasted computations, resulting in faster jobs turnaround times. In particular one of our strategies has specifically treated the interference failures which are the main reason of resources volatility in a desktop grid or global computing context. Our evaluation results showed marginal improvements on valuable exploitation of the system and more research is needed for optimizations of this technique.

Indeed, the complication of real large-scale experimentation along with the big number of parameters that influenced the function of the system did not let us to study all different factors in depth. Particularly more research is needed for size of the checkpoints along with the respective time to generate them and how this influences the results of our strategies. A future version of our implementation will take into account the work in [175] in order to predict the time that an application needs to checkpoint itself giving the capability to the scheduler to dynamically set the grace-time delay.

Our approach initially was designed and implemented for grids with homogeneous clusters so jobs could be restarted upon any cluster without migration limitations. This assumption simplified our initial implementation since we did not deal with architecture dependency issues. The migration possibilities could be also studied in a context of shared checkpoints between clusters. This approach could provide better system utilization but with an impact to jobs waiting times.

Finally a big part of the contributions of this chapter lies on the experimentation methodology. The real-scale experimentation of a real grid platform (DAS2) upon a controlled infrastructure (Grid5000) is something that was not possible before. Moreover the use of a real astrophysical application related with the fact that all the computations performed for the experimentation phase were stored and utilized for scientific purposes made our experimental testbed an interesting research tool. Nevertheless we argue that a real-scale experimentation method for this kind of experiments has to be preceded by simulation results in order to spot the interesting paths of research. This will help avoid wasting computational cycles for experimenting with approaches that have limited performance.

Chapter 5

Improving resources exploitation with Malleability techniques

One of the latest evolutions of current parallel architectures is the use of multicore processors. Resource and Job Management Systems (RJMS) need to be sufficiently equipped in order to be capable to treat this new hierarchy of resources. Limited treatment of this affinity of resources results in inefficient utilization of the system. In the same time, the high complexity of workloads, with variations in resources demands and execution times may result in bad scheduling performance producing long waiting jobs, big turnaround times and large underutilization periods. In such dynamic scenario, achieving a good resources utilization is a challenge. The scheduler of the resource and job management system is attributed the difficult task of adapting itself upon the high dynamicity of users' workloads.

In order to treat inefficiency and underutilization in a similar context, the RJMS could provide the support of applications that can adapt themselves on resources availabilities. According to Feitelson and Rudolph [13], parallel applications can be: rigid, moldable, malleable or evolving. We are especially interested in malleable jobs because they can adapt themselves to resources with dynamic availability, and thus be able to follow the RJMS decisions for schedule adaptation. However, the support of malleable jobs demand a special treatment, from the RJMS side, with more complex allocation and scheduling procedures. In particular, specific communication and negotiation procedures need to be developed, which will enable the on-the-fly adaptation of the application. New scheduling policies need to be defined in order to take into account recources dynamicity. In this study, we examine the complexity of treating malleable jobs upon the RJMS while observing improvements on resources utilization. Furthermore, we extend a flexible RJMS,

so as to provide the support of malleable applications.

Most of todays parallel applications belong to the class of rigid type, which are programmed for a specific constant number of resources. The complexity of programming a malleable application or even reprogramming a normal rigid parallel application into malleable, implies important difficulties that have been a real challenge for programmers. In more detail, applications that use the MPI-2 implementation need to use MPI_comm_spawn function for dynamic process creation in case of application growing and adapted fault-tolerance mechanisms in case of shrinking. Hence, due to programming complexities, the number of existing malleable applications is actually limited. Indeed, as shown in following chapter, most of related works try to solve those programming difficulties by proposing a co-design of the programming environment and the runtime system. However, it seems that even if those systems solve some difficulties, their utilization is heavy and complex. Therefore, in an effort to allow a bigger number of applications to take advantage of malleability features and at the same time to enable the RJMS with the possibility of better resources exploitation, we moved a step further. We have implemented a simple transparent technique for automatic expanding and folding of rigid applications, upon multicore architectures.

Our study uses two approaches to provide malleability: dynamic MPI and dynamic CPUSET mapping. The first one uses a dynamic MPI application, able to adapt itself to the available cluster nodes. Dynamic MPI (malleable) applications are developed by employing MPI-2 dynamic process creation and fault tolerance mechanisms. The second approach is well-adapted to multicore architectures and allows any parallel job to exploit malleability through a -transparent to the application- fine control and manipulation of the available cores.

The support of those two malleability approaches is implemented upon the resource and job management system OAR [79]. Our prototype was tested in real-scale upon the controlled platform Grid5000 uing real production workload traces

5.1 Background and Related Work

In this section we describe some Background information in order to specify the context of our work and we analyze the Related works on the field of malleability and its support by the runtime system. We examine the subject from 3 different perspectives. The first one has to do with the applications and the workloads, along with the programming models used to implement adaptive applications. The second one is related to the research conducted in the field of Scheduling of

Who decides	When decided	
	At submission	During Execution
Application	Rigid	Evolving
System	Moldable	Malleable

Table 5.1: Classification of parallel applications

adaptive applications along with the approaches for support upon RJMS. Finally, the third category examines previous work of the communication protocols between applications and runtime systems.

5.1.1 Dynamic Applications and Programming

In the context of resource and job management systems, parallel applications are represented as jobs, which have a running time `t` and a number of processors `p`. Feitelson and Rudolph [13] proposed four job classes based on the relation of who is responsible for the decisions and which moment these decisions can take place 5.1. *Rigid* jobs require a fixed `p`, specified by the users, for a certain time `t`. In *moldable*, the RJMS chooses a specific `p` followed with its related `t` from a range of choices given by the users. The decision is made at start time and the job adapts to `p` that will not change until the end of the execution. In *evolving*, the `p` may change on-the-fly according to the jobs requests. This adaptation must be satisfied by the RJMS in order to avoid crashes. In *malleable*, `p` may also change on-the-fly, but the changes are required by the RJMS, according to resources availabilities and the schedulers decisions.

The structure of dynamic applications is different from that of rigid applications. They need to reconfigure themeselves when additional resources are consumed or idle resources are released. The reconfiguration may require data redistribution, creation of new processes or deletion of existing ones, etc.

The Message Passing Interface (MPI) is currently the most dominant model for programming parallel computers today. The static model of MPI-1 means the number of tasks is fixed at job launch time. The MPI-2 specification added support for dynamic process management which allowed MPI applications to create and communicate with new processes, thus providing a new paradigm for programming MPI applications. Various implementations of the MPI-2 standard exist that support the dynamic process management LAM-MPI, MVAPICH2, MPICH2, OpenMPI.

There are a lot of scientific and engineering applications which work with large input data and

are computationally intensive. The total amount of computation for these applications does not vary from execution to execution for the same input data. Currently these applications are implemented as rigid or moldable. These applications are good candidates for conversion to malleable. As far as our knowledge no real malleable application was found in the literature. Hence, for the sake of our experiments we have reprogrammed a traditionally rigid application into malleable one (chapter 2 section 2.5). Nevertheless, as described on chapter 4, a big number of scientific applications that can be considered malleable ones is the bag-of-tasks which are embarassingly parallel applications.

Examples of evolving applications are parallel applications where computational workload varies during the execution due to the nature of the problem, the employed algorithm and an unpredictable non-uniform distribution of input data. An example of evolving applications can be considered the parallel Fast Multipole algorithm for N-body simulation [176]. This problem has been used in different areas of science like astrophysics, molecular checmistry, biophysics, etc. The total computational workload of these applications does not vary from execution to execution for the same input data but the computational resources requirements change during a single run.

On the other hand, there exist evolving applications where computational workload varies from execution to execution for the same input data. The most important example in the category is weather prediction. Even if the traditional weather forecasting systems are static in nature, the researchers are working on the next generation which will adapt dynamically both in time and space[177]. Due to unpredictable data that occur during their execution the amount of computation may be in constant change.

Malleable applications provide difficulties in their programming but may pay off those complexities because they can atain smaller waiting and turnaround times on the queues of the RJMS scheduler. Moreover, their adaptative nature may result in more efficient system utilization. On the other hand evolving applications not only poses difficulties on their programming but their use in mixed workloads provides additional complexities, to the scheduler, due to their opportunistic behaviour. The RJMS that supports evolving applications needs to be equiped with specific preemption mechanisms in order to be able to service their requests. Our study is focused only on the class of malleable applications.

5.1.2 Resource Management and Dynamicity

There is a growing interest to offer malleability in practice, because it can improve both resource utilization and response time [13, 178, 179, 180, 181]. Nevertheless, very few actual implementations have been proposed in the community.

A good approach towards the support of malleable jobs can be achieved by a co-design of runtime-system and programming environment [13]. This assertion can be observed in some practical malleability initiatives like PCM and Dynaco.

PCM (Process Checkpointing and Migration) [182, 183, 184] uses migration to implement split and merge operations reconfiguring application according to resources availability. PCM demands an instrumentation of MPI code, using its API and specifying which data structures are involved in malleable operations. It interacts with a specific framework, IOS (Internet Operating System), performing like a resource manager, which is composed by an agents network to profile informations about resources and applications.

In the same way, Dynaco [185] enables MPI applications to spawn new processes when resources become available or stop them when resources are announced to disappear. To provide malleable features, Dynaco must know the adaptive decision procedures, a description of problems in the planning of adaptation, and the implementation of adaptive actions. Also, the programmer must include *points* on its source code identifying safe states to perform adaptive actions ensuring application correctness. Dynaco has its own resource manager, Koala, providing data and processor co-allocation, resource monitoring and fault tolerance.

Faucets [186] is an Adaptive Job Scheduler aiming at maximize the utility metric of a cluster and ensure that quality of services requirements are met. Malleability is provided by Adaptive MPI (AMPI), implement upon Charm++, which extends MPI supporting processor virtualization and automatic load balancing through virtual processes migration. Job's QoS requirements taken into account to schedule jobs include the minimum and maximum number of processors that a job can run on, an estimated number of CPU-seconds required, and a deadline before which it must be completed. After take the appropriate decision the scheduler send a bit-vector to processors, which contains informatios to shrink or expand jobs. A bit-vector handle sets the bit-vector in the load-balancer, and in the next load-balancing cycle will use the newly information to reconfigure the job. In this way the shrink and expand mechanism has a latency of one load-balancing cycle.

Excluding initiatives with their own systems to provide resource management like the previously introduced, and according to our up-to-date knowledge, there is no existing implementation of malleable jobs support upon a generic resource and job management system since it is a rather

complicated task. Nevertheless, some previous work have studied specific prototypes for the direct support of malleable jobs [181, 180] with different constraints. For instance, Utrera et al. [180] proposes: a virtual malleability combining moldability (to decide the number of processes at start time), folding (to make the jobs malleable) and Co-Scheduling (to share resources). It uses Co-Scheduling to share processors after a job folding and migration to use the new processors after a job expansion. Our work also explores a similar folding technique, with dynamic CPUSETs mapping in multicore machines. Further, its Launcher component is a user-level queuing system, that decides the optimal number of processes and folding times to an arriving job according to its size, the resources available and the running and waiting jobs. A CPU Manager schedules the job upon the resources after its launching and monitorates jobs verifying if some is arriving or finishing, and if necessary, it redistributes the processors. All informations needed by Launcher or CPUM are shared through control structures.

In [181], an Application Parallelism Manager (APM) provides dynamic online scheduling with malleable jobs upon shared memory architectures (SMP). The approach combined the advantages of time and space sharing scheduling. It achieved a 100% system utilization and a direct response time. Nevertheless the system was not directly adaptable to distributed-memory machines with message-passing and this work, as far as we know, has not evolved in a real implementation of the prototype.

As far as our knowledge, concerning malleable applications, two commercial resource and job management systems support this kind of dynamicity, and these are LSF [187], described on 3.4.1 and Moab [188], described on 3.4.1. Nevertheless no specific publication has been found that experiments or evaluates those features.

Finally, Ghafoor in his PhD thesis [15] has made an analytical study of adaptive applications and his work included the implementation of a prototype resource management system that supports mixed workloads with malleable and rigid jobs.

5.1.3 RJMS and Applications Communication protocols

In this section we describe the communication and negotiation protocols between the RJMS and the applications. In the traditional job-scheduling paradigm, once a job starts executing, no communication takes place. In the case of a parallel system that supports malleability the RJMS needs to inform the application for its exact resources availabilities along with the request for offering or retrieving resources.

MPI applications use process managers to launch them as well as get information such as their

rank, the size of the job, etc. In the traditional static MPI paradigm the communication takes place only in the initialization of the execution. However, in the dynamic MPI-2 paradigm communications may take place during the execution as well. Hence, to support adaptive applications, specific libraries that support dynamic process management are needed. It seems that every MPI-2 compliant implementation of MPI standard posseses its own specific communication library and runtime support.

The MPICH2 implementation of MPI standard, specified an interface called the process management interface (PMI) [189] that is a set of functions that MPICH2 internals (or the internals of other parallel programming models) can use to get such information from the process manager. However, this specification did not a defined protocol on how the client-side part of the PMI would talk to the process manager. Thus, many groups implemented their own PMI library (with common inteface). For example apart the default PMI library of MPICH2, Slurm RJMS provides its own PMI library. In order to answer to scalability issues of modern platforms PMI version 1 is not sufficient enough and PMI-2 [190] is currently under construction. One of the new updates is the improvement of dynamic process management.

In addition, the OpenMPI implementation uses its own Open Run Time Environment ORTE [191] and LA-MPI uses [192]. Hence from the point of view of the RJMS that wants to provide support for every MPI implementation it needs to support all the different communication protocols. For example Slurm RJMS has implemented its own PMI library in order to communicate with all different MPI implementation based on MPICH it supports MVAPICH2, MPICH2 and others. Hence, a most general runtime support is needed and it seems that a possible solution could be STCI [193] which is an ongoing project that aims to provide a unified platform for the implementation of a wide variety of tools for high performance computing. It proposes basic services to provide all needed mechanisms and policies for the implementation of the management of communications in a scalable manner or other scalable tools, like debuggers or the management of user session and security issues.

In our study we have used LAM [194] MPI implementation which does not provide its own communication library. In order to communicate with the applications during execution we have implemented our own communication and negotiation protocols based on simple system calls and file exchanges.

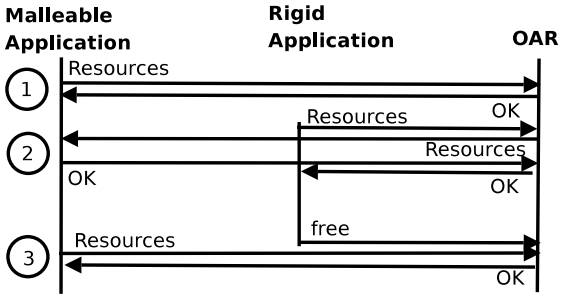


Figure 5.1: Single malleable application scenario. (1) Starting time; (2) shrinking operation; (3) growing operation.

5.1.4 Scheduling with Dynamic Applications

Scheduling algorithms for dynamic applications are more complex than those for rigid applications. The scheduler has to respond as fast as possible to the demands not only of the queued applications, but of the running ones as well. The category of evolving applications need a special treatment concerning its scheduling. On the applications requests if enough resources are not available the scheduler needs to perform preemption mechanisms (suspend/resume or checkpoint/restart) in order to respond to the demand. On the other hand malleable applications can ease the difficulties of the scheduler since they can adapt on resources availabilities and on schedulers demands. Scheduling in a dynamic parallel system is a multi-step procedure that involves the scheduling, the communication and negotiation with running applications and the re-computation of the schedule.

There are relevant theoretical results describing requirements, execution scheme and algorithms to efficiently schedule malleable applications [14]. Scheduling algorithms that have been developed so far vary in their objectives and approaches selected. A thorough survey of related work on scheduling of adaptive applications can be found on [15].

Equipartition scheduling policy distributes the available resources equally upon all running malleable jobs [185, 181] as shown in figure 5.2. This approach has some issues as: *(i)* the amount of available resources is not divisible by the number of malleable jobs running – Buisson at al. [185] proposes to give the remainder resources to the least recently started job; *(ii)* If all jobs receive the same amount of resources, smaller ones will finish earlier probably causing a new resource distribution. Remaining unallocated resources are shared equally, respecting the maximum limitation of each job. Experiments with actual applications and simulation showed that both system utilization and mean response time improves with reconfigurable scheduling when compared to static

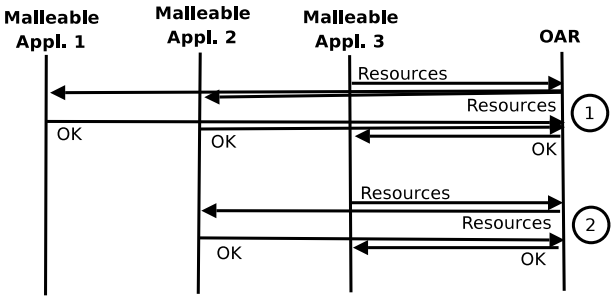


Figure 5.2: Multiple malleable application scenario. (1) Equipartition; (2) FPSMA

scheduling consisted by moldable jobs only.

A similar algorithm is used in [15] and [178]. Each arriving job specifies the minimum and maximum number of resources it can use. When a new job arrives, the scheduler recalculates the number of resources allocated to each running job and redistributes them in a way that all jobs are allocated their minimum number they can use.

Favour Previosly Started Malleable Applications FPSMA [185] policy favours previosly started malleable jobs, as shown in figure 5.2. It starts growing from the earliest started malleable job and shrinking from the lastest started one. In this approach, if the lastest job has suficient resources shrinkable to attend an arriving job, it perform the resource scheduling requiring less communications than the Equipartition policy. The key issue is determine the initial requirement of an malleable application. Whether malleable jobs can require a especific amount of resources at start time, it is possible perform as communication as performed by Equipartition, in consequence performing many shrinking operations.

Further, the *Maximizing the Accumulated Current Speedup* $maxS*$ [181] takes the scalability of jobs into account to increase the system usage efficiency. It considers the sum of all jobs speedups and the diferencial speedup, which is the speedup difference when a job going from $p-1$ to p resources. The job with the largest differential speedup gets the resource assigned. A new diferencial speedup is calculated considering the resource assigned and this procedure is repeated until all resources are distributed. This approach is hard to illustrate because it depends on speedup and differential speedup mesures, which can vary according to the application nature. Also, it is dependent of speedup estimations and the knowledge of applications paramenters: execution time, number of resources by job, communication overhead and so on. This approach schedules one resource at time, this can induce a malleable application to perform many malleable

operations (growing or shrinking) one by one.

In this study we have focused on a context where the scheduler allows only one malleable job at a time, mixed with rigid jobs (as shown in figure 5.1). This choice was made, in order to be able to measure the overheads of the communication protocols along with the advantages upon system utilization with the two different approaches for malleability. However studies are under way for supporting mixed workloads of rigid, moldable and malleable jobs.

5.2 Malleability techniques: Implementation and Experimental Testbed

The main goal of our approach was to provide malleability, for optimization of the systems utilization, without interfeering with the normal function of the resource and job management system. The flexible and modular architecture of OAR [79] made it the most suited RJMS for implementing our prototype. In this section we provide the architecture of this prototype for malleability and its implementation as a wrapper around OAR. A description of the experimentation procedure is also provided.

5.2.1 Implementation upon OAR resource manager

The prototype for malleability was designed as an external module to OAR that communicates with OAR through normal user commands (`oarsub` for submission and `oarstat` for resource discovery). To keep the prototype non intrusive to OAR, we exploit the capabilities of *Best Effort jobs* which are able to harness the idle resources of a cluster, with no interference upon its normal function but also no guarantees for task termination. However, we wanted to provide at least a minimum guarantee for the malleable application to terminate itself, so we considered that a normal job, with minimum resources requirements, should be submitted as well in order to make sure that the application will succesfully terminate. In addition, the specific resources allocated by this normal job will be utilized, by the prototype, to host the particular modules that will manage the dynamic operation of the application (*Malleability worker*). Figure 5.3 provides a graphical representation of the prototype which is represented by the different modules with red letters.

The prototype implementation consists of the following parts:

- the *Malleability automaton* for submission of jobs and decision making

- a *Normal job*, for management of dynamicity and guarantees for termination
- the *Malleability worker* which recides on the resources allocated by the normal job and is responsible for the management of dynamic operations
- *Best Effort jobs*, for the adaptability and non intrusiveness to the cluster
- a *Resource Discovery command*, which provides the current and near-future available resources
- a *Communication Protocol*, through sockets and file exchanges

Hence, to provide malleability using OAR we consider that a malleable job is constructed by a rigid and a Best Effort part. The rigid part stays always intact so that it can guarantee the job completeness. In the same time, the Best Effort part is responsible for the flexibility of the job. Hence, using the resource discovery command, which informs the application for the variations on the resources availability can lead to two actions: Best Effort jobs can be either *killed* meaning application shrinking or further *submitted*, allowing the application growing.

As we will see on the following sections each malleability technique has each own particularities and requirements but both are based on this same design. The differences reside on the number of allocated resources that the normal job will use along with the functions of the malleability workers in each case.

5.2.2 Experimentation through controlled platform and real traces

The experimentation that take place to validate our approaches and evaluate the behaviour of the system for each technique is based upon the methodology presented in chapter 2. For the particular experiments we have made use of Bordereau cluster in Bordeaux site of Grid5000 platform. The cluster is composed by nodes of AMD Opteron 2218 processor (DualCPU-DualCORE with 2.6 GHz / 2 MB L2 cache / 800 MHz) with 4GB Memory and Gigabit Ethernet network. During the evaluation procedure of the malleability prototypes we provide a first series of experiments to validate the requirements and choices for each technique and measure the possible overheads or side-effects of each method.

A second series of experiments was performed to measure the system utilization improvements when using malleability techniques and compare it with a specific type of moldable approach. For these experiments we use workload log of a real production system (DAS2 [5]). In our testbed

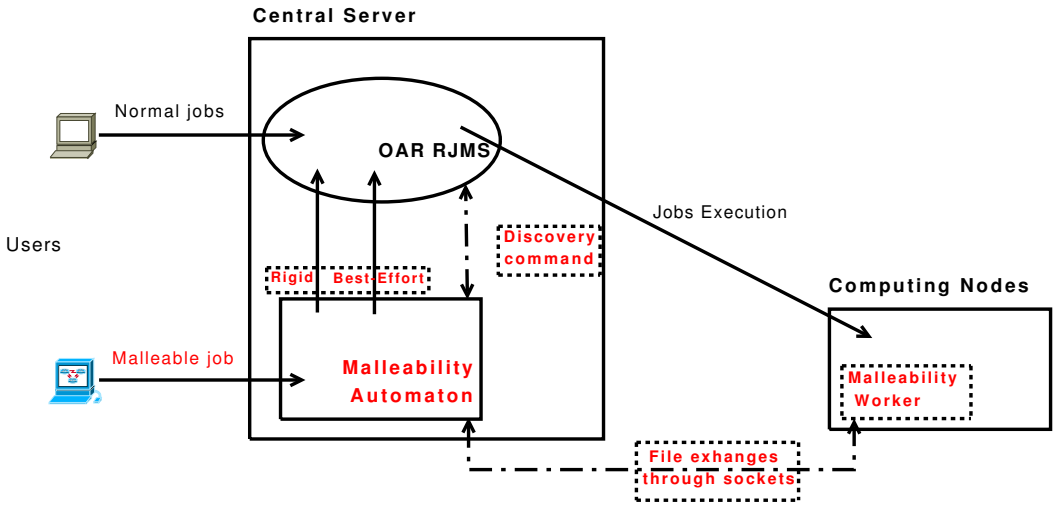


Figure 5.3: Malleability architecture upon OAR

we deploy this trace as the default rigid workload of the system and by enabling the submission of one malleable job per time we calculate the optimization in system utilization. Only when the malleable job has finished execution a new one can be submitted. We compare the optimization in system utilization to the one that we could have in case of using a *moldable-besteffort* job which cannot change its allocation during the execution.

In our context we are interested in a flexible environment where the dynamic jobs should be the less intrusive the possible for the normal functionality of the system. That's why we do not compare the malleability techniques to a pure moldable jobs behaviour which would be intrusive to the normal workload but to a *moldable-besteffort* job submission. In this context the *moldable-besteffort* job would be killed in case of need of resources by a rigid job and a new *moldable-besteffort* job, adapted to the new availabilities, would be submitted. The *moldable-besteffort* job is exactly the context of jobs used for global computing or desktop grids (usually bag-of-tasks applications as studied on chapter 4)

The advantage of *moldable-besteffort* jobs is that they have no side-effects on the default system utilization and no impact on the jobs waiting times of the normal workload. On the other side the *malleable* jobs provide a rigid part which is expected to impose a particular overhead and an increase of the jobs waiting times. Therefore, our evaluation also considers the impact of the malleability techniques on jobs waiting times.

During our experiments we have made use of Mandelbrot application reprogrammed to malleable one as described on section 2.3.4 for the Dynamic MPI technique and the CG and BT (class

C) kernel and application of the NAS parallel benchmark suite [66] for the Dynamic CPUSET technique.

5.3 Malleability on Clusters with Dynamic MPI

The malleable behaviour of the Dynamic MPI mechanism is defined by the respective implemented modules upon OAR and the adaptive capabilities of the particular malleable MPI application. The application performs growing and shrinking operations to adapt itself to the variations in the availability of the resources. This section provides the particularities of the implementation of the technique upon OAR along with a study for performance evaluation. This work was made in collaboration with Marcia Cristina Cera and her research group GPPD of Parallel and Distributed Processing Group from UFRGS university in Brazil.

5.3.1 Dynamic MPI mechanism Requirements

The implementation of the *Dynamic MPI* technique upon OAR followed the general paradigm presented on the previous section. Figure 5.4 show the graphical representation of the architecture of this malleability technique. We can see that the *malleability worker* of figure 5.3 is represented by a particular module `D-MPI manager` which is responsible to deal with the requirements of the malleable MPI applications.

Indeed to manage each different case of dynamicity (growing or shrinking) the `D-MPI manager` module needs to provide specific techniques. Those techniques need to be adapted on requirements of the particular MPI implementation used for the programming of the MPI application. In our case was LAM/MPI[1]. As previously explained, in our experiments, a malleable job is composed by a rigid part (normal job) and Best Effort jobs. In the context of dynamic MPI, the rigid part is composed by the minimum unit manageable in LAM/MPI, which is one node. The Best Effort part is as large as the amount of resources available, which is determined by the resource discovery command.

When new idle resources become available the discovery command notifies the Malleability automaton which issues a growing operation. In growing, a part of the applications' workload is destined to the newly available resources through the mechanism of MPI process spawning, which is a feature of the MPI-2 norm (MPI_Comm_spawn and correlated primitives) [195]. This

[1] $http://www.lam-mpi.org/$

is possible by the socket communication between the `Malleability automaton` (Central Server) and the `D-MPI Manager` (computing node).

In shrinking, processes running on resources which have been demanded by the resource manager (for higher priority jobs) must be safely finalized. This interruption requires a specific procedure in order to prevent application crash. In our context We adopted a simple one which is to identify the particular tasks and restart them in the future. It is not optimal, but ensures application results correctness.To ensure that this operation will be safely performed, a grace time delay is applied upon OAR, before the killing of the Best Effort jobs. Grace time delay represents the amount of the time that the OAR system waits before destinate the resources to another job, ensuring that they are free. The same solution was also presented on the previous chapter. Figure 5.5 illustrates a malleable job upon 4 nodes with 4 cores performing growing and shrinking operations.

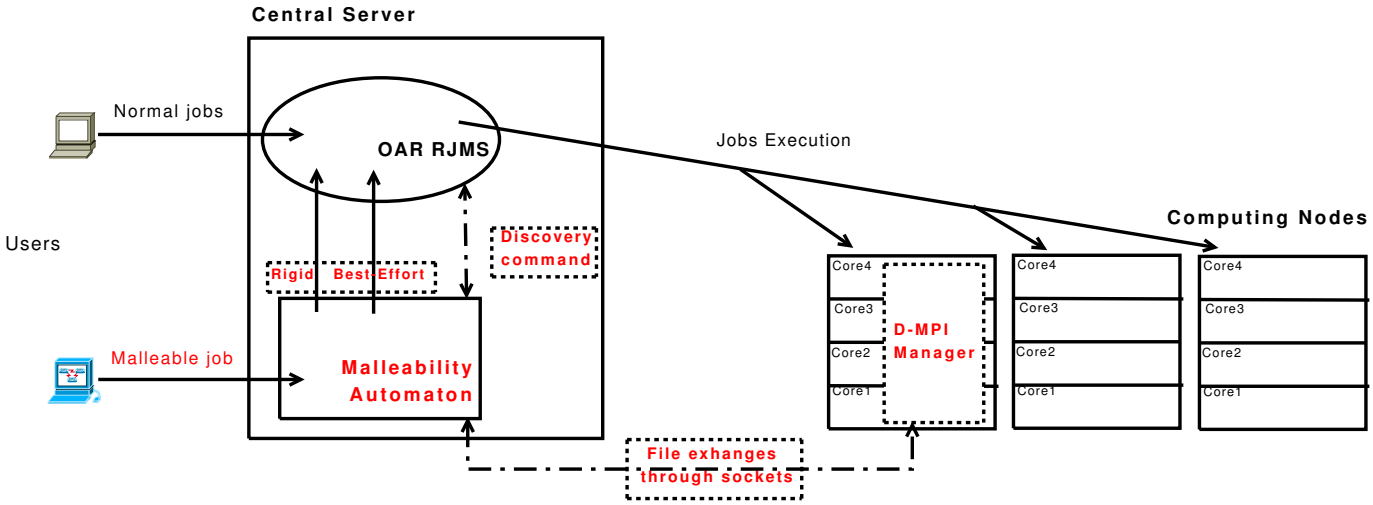


Figure 5.4: Dynamic MPI architecture

Entering more into the details of the dynamicity upon MPI applications; the malleability operations, growing and shrinking, are handled by the `D-MPI Manager` 5.4. This module interacts with the `Malleability automaton` to receive information about variations in the amount of resources available to dynamic MPI applications. According to these informations, `D-MPI Manager` triggers the appropriate dynamic action enabling the adaptation of the application to the availability of the resources. As part of the growing operation, it ensures that spawning processes will be placed into the new resources. The library intercepts the MPI_Comm_spawn calls and sets

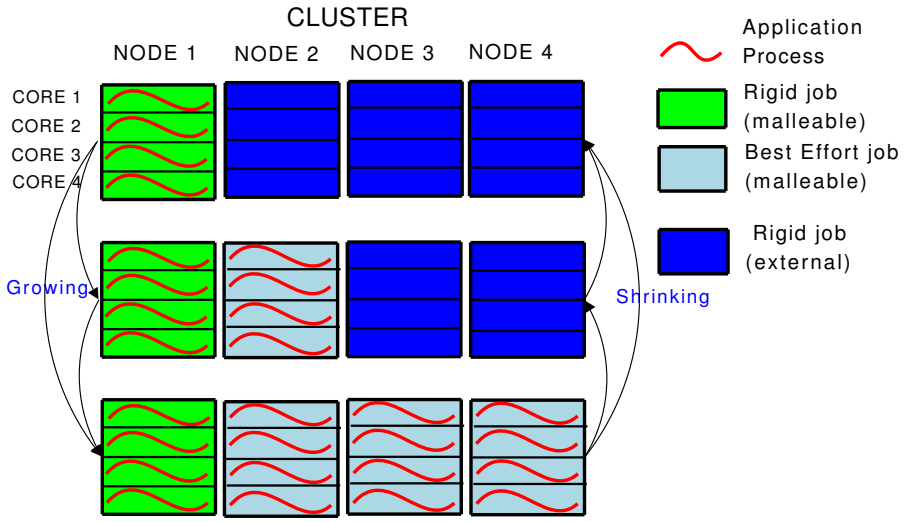


Figure 5.5: Dynamic MPI application performing growing and shrinking upon 4 participating nodes.

the physical location of the new processes according to the informations provided by the automaton. Previous work of our colleagues in Brazil, did not consider interactions with the resource manager and proposed one of two scheduling policies for the dynamicity of jobs: Either a simple Round-Robin (standard) or workload-based (based on the less overloaded resource) [196].

In the experiments of this chapter, `D-MPI Manager` and dynamic MPI application are implemented with LAM/MPI[2] distribution. This MPI distribution offers a stable implementation of dynamic process creation and ways to manage dynamic resources. This last feature is provided by two commands: `lamgrow` to increase and `lamshrink` to decrease the amount of nodes available in LAM/MPI network (LAM/MPI applications run up a known set of resources, which begins by `lamboot` before application starting time and ends by `lamhalt` after application execution). Note that LAM/MPI enables the management of nodes, which can be composed by many cores, but cannot manage isolated cores. The `lamgrow` and `lamshrink` commands are always called by `D-MPI Manager` when some change is announced by the `Malleability automaton`.

5.3.2 Evaluating Dynamic MPI technique

In this first part of evaluation of the technique our goal is to validate our choices, measure the speedup of a malleable application when using the Dynamic MPI malleability technique. We have

[2] $http://www.lam-mpi.org/$

made use of the Mandelbrot malleable application and provoked triggered growing and shrinking in order to observe possible side-effects of this technique. We then calculated the theoretic speedup and performed comparisons between our experimental and mathematical results.

For these experiments we used the maximum of 64 cores, coming from 16 nodes of the used cluster described in section 5.2.2. Figures 5.6 and 5.7, show the execution time of the dynamic MPI application performing growing and shrinking operations respectively. In the growing, the application begins with the amount of cores shown in x axis (representing 25%, 50% and 75% of the total number of cores) and grows until 64 cores. In shrinking, it starts with 64 and shrinks until the x values of cores. We made two experiments: *(i)* a dynamic event is performed **at** a time limit, and *(ii)* dynamic events are gradually performed **until** a time limit. Time limit is defined as 25%, 50% or 75% of the parallel reference time. Reference time is always the execution time with the initial number of cores, i.e., x to growing and 64 to shrinking.

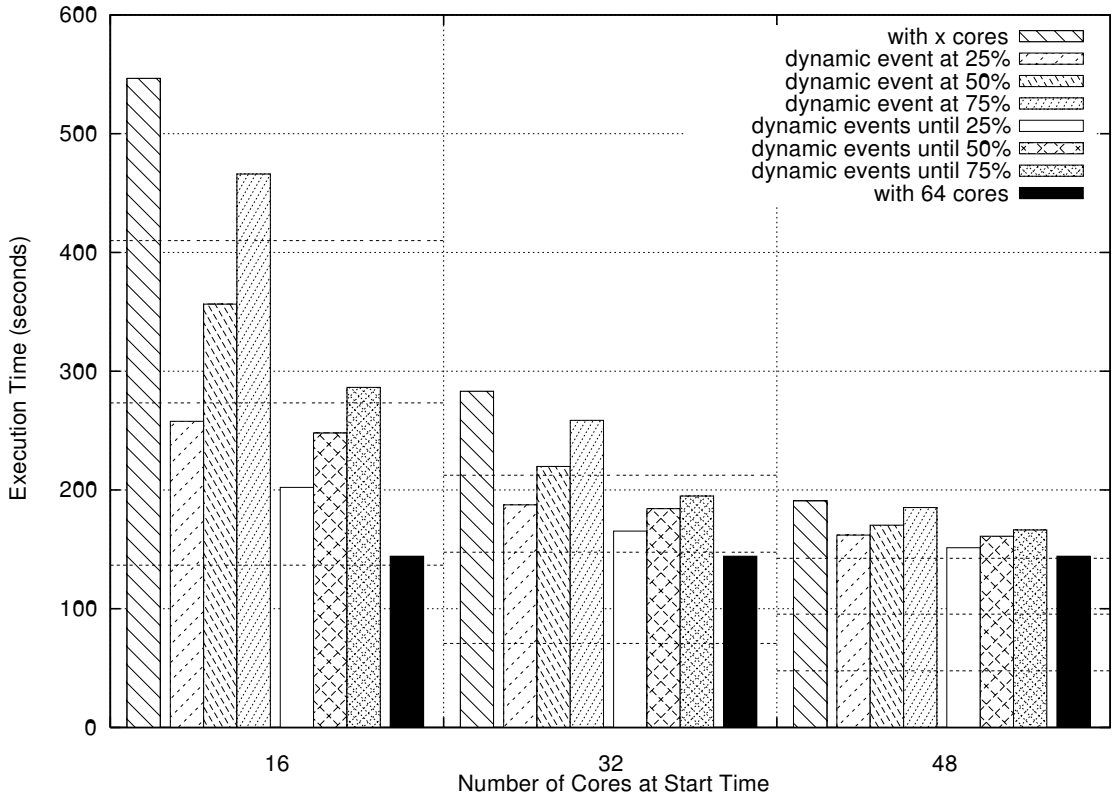


Figure 5.6: Growing in dynamic MPI application: execution time VS number of cores at starting time.

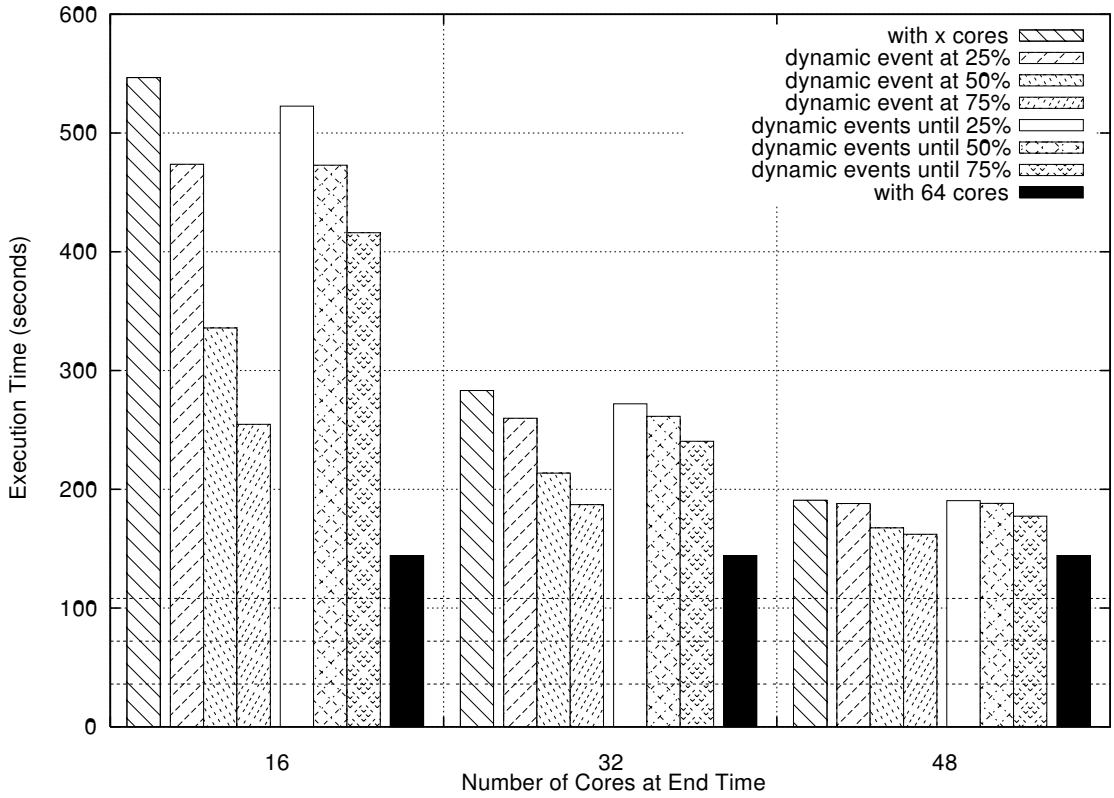


Figure 5.7: Shrinking in dynamic MPI application: execution time VS number of cores at ending time.

Let p_r be the number of cores on which a reference time t_r has been measured. Thus, the parallel reference application performs $W = t_r \times p_r$ operations. The application is progressively run on more than p_r cores:

- In the first experiment (dynamic events at a time limit) it is run during $\alpha t_r (0 \leq \alpha \leq 1)$ seconds on p_r cores, and then on p cores;
- In the second experiment (dynamic events until a time limit) it is run during $\alpha t_r (0 \leq \alpha \leq 1)$ seconds on a gradual increasing or decreasing number of cores, until reaching p cores. Then it is run until completion, without anymore changes. The number of cores increases or decreases at regular timesteps of c units. In fact, $c = 4$, since the `lamgrow` command only enables to add a whole node (2 CPUs with 2 cores) to LAM/MPI network. The number of gradual timesteps is therefore $\frac{p-p_r}{c}$, and each one lasts $\delta = \frac{\alpha t_r}{\frac{p-p_r}{c}}$ sec.

Ideal speedup in the first experiment. The execution on p_r cores has duration αt_r. At this point,

$(1-\alpha)W$ operations remain to be performed, by p cores. Ideally, this second phase runs in time $(1-\alpha)W/p$. Therefore, the total parallel time in this case is $t_p = \alpha t_r + (1-\alpha)t_r p_r/p$, and the speedup t_r/t_p is:

$$S = \frac{1}{\frac{p_r}{p}(1-\alpha)+\alpha} \tag{5.1}$$

Ideal speedup in the second experiment. As in the previous case, the number of operations performed during the first αt_r seconds, can be computed, to obtain the parallel time of the second part of the execution on p cores. At the ith timestep ($i = 0...\frac{p-p_r}{c} - 1$), the program is run with $p_r + ci$ cores in time δ sec. Therefore, the number of operations n_i that are executed is $\delta(p_r + ci)$. When all the p cores are available (i.e. at time αt_r), $\sum_{i=0}^{(p-p_r)/c-1} n_i$ operations have been run, leaving $W - \sum_{i=0}^{(p-p_r)/c-1} n_i$ to be run by p cores. Thus, the second phase has duration:

$$\frac{t_r p_r - \sum_{i=0}^{(p-p_r)/c-1} \delta(p_r + ci)}{p}, \tag{5.2}$$

and the total execution time in this second experiment is therefore:

$$t_p = \alpha t_r + \frac{t_r p_r - \sum_{i=0}^{(p-p_r)/c-1} \delta(p_r + ci)}{p}. \tag{5.3}$$

Besides,

$$\sum_{i=0}^{(p-p_r)/c-1} \delta(p_r + ci) = \delta p_r \frac{p-p_r}{c} + c\delta \sum_{i=0}^{\frac{p-p_r}{c}-1} i = \delta p_r \frac{p-p_r}{c} + c\delta \frac{(\frac{p-p_r}{c}-1)\frac{p-p_r}{c}}{2}. \tag{5.4}$$

And since $\delta = \frac{\alpha t_r}{\frac{p-p_r}{c}}$, the latter equation yields:

$$\sum_{i=0}^{(p-p_r)/c-1} \delta(p_r + ci) = \alpha t_r p_r + \frac{\alpha t_r}{2}(p - p_r - c). \tag{5.5}$$

Therefore, the total parallel time in this case is:

$$t_p = \alpha t_r + \frac{t_r p_r - \alpha p_r t_r - \frac{\alpha t_r}{2}(p - p_r - c)}{p} = \frac{t_r}{2p}(2p_r + \alpha(p - p_r + c)). \tag{5.6}$$

And the parallel speedup t_r/t_p is:

Table 5.2: Speedup of the dynamic MPI application.

Growing							**Shrinking**						
p_r	p	α	S	$S(eq.5.1)$	S	$S(eq.5.7)$	p_r	p	α	S	$S(eq.5.1)$	S	$S(eq.5.7)$
16	64	0.25	2.12	2.29	2.70	2.84	64	16	0.25	0.30	0.31	0.28	0.27
		0.50	1.53	1.60	2.20	2.21			0.50	0.43	0.40	0.30	0.30
		0.75	1.17	1.23	1.91	1.80			0.75	0.57	0.57	0.35	0.34
32	64	0.25	1.51	1.60	1.71	1.75	64	32	0.25	0.55	0.57	0.53	0.53
		0.50	1.29	1.33	1.54	1.56			0.50	0.68	0.67	0.55	0.56
		0.75	1.09	1.14	1.45	1.41			0.75	0.77	0.80	0.60	0.60
48	64	0.25	1.18	1.23	1.26	1.27	64	48	0.25	0.77	0.80	0.76	0.77
		0.50	1.12	1.14	1.18	1.21			0.50	0.86	0.86	0.77	0.79
		0.75	1.03	1.07	1.15	1.15			0.75	0.89	0.92	0.81	0.81

$$S = \frac{2p}{2p_r + \alpha(p - p_r + c)}. \tag{5.7}$$

Table 5.2 shows the speedups, in which S is the speedup in practice (t_r/t_p), $S(eq.5.1)$ is the ideal speedup to the first experiment using the Equation 5.1, and $S(eq.5.7)$ to the second one using the Equation 5.7. In growing, practical speedups are slightly lower than the ideal ones, representing the overhead to perform the spawning of new processes. As Lepère et al. [14] the standard behavior in on-the-fly resources addition is that such addition cannot increases the application execution time. In the Table 5.2, we observe that growing speedups are always greater than 1, meaning that the new cores could improve application performance decreasing its execution time as expected. In shrinking, all speedups values are lower than 1, as the exclusion of nodes decreases the performance. In this case, practical speedup is quite similar to the theoretical one and the variations become from the execution of the procedures to avoid application crash. Summing up, the speedups show that our dynamic MPI application is able to perform upon resources with a dynamic availability.

5.4 Malleability on Multicore Nodes with Dynamic CPUSET Mapping

The malleable behaviour of the Dynamic CPUSET Mapping technique is based on an internal Linux Kernel mechanism for task confinement upon paticular cores of a computing node of the

cluster. This mechanism provides the necessary adaptability of executed tasks for the malleability technique. The particular implemented modules of the prototype perform folding or expanding of the application upon single multicore nodes using exactly the resources that are unutilized by the normal workload of the cluster. This section provides the particularities of the implementation of the technique upon OAR along with a study for possible sideffects.

5.4.1 Dynamic CPUSETs Mapping Requirements

CPUSET [82] are lightweight objects in the Linux kernel that enable users to partition their multiprocessor machine by creating execution areas, and confine the execution of the application tasks. They are used by the resource management systems to provide CPU binding to tasks and cleaner process deletion when the job is finished.

Dynamic CPUSET mapping technique is the on-the-fly manipulation of the amount of cores per node allocated to an application. This technique is inspired by the latest advances in multiprocessing (increasing number of cores per single CPU) and the flexibility of the CPUSET objects to bind tasks upon cores of multicore architectures. *Dynamic CPUSET mapping* allows expanding or folding of the application upon the same node. In more detail, if we have multiple processes of a MPI application executing upon a multicore node, then those processes can be executed upon one or more cores of the same node.

It is a prototype system level technique which can provide a level of malleability upon any parallel application and it is completely independent of the application source code.The only restriction is that the malleability is performed separately and individually upon each node. In contrast with the *Dynamic MPI* technique there are no new processes of the application that are spawned. All the application processes are initiated in the beggining but they are bound to only the cores that are initially available. To take full advantage of the malleability possibilities, the number of processes started upon a node should be equal or larger than the number of cores per node. For instance, a 16 processes MPI application should be ideally initiated upon 4 nodes of a 4 cores-per-node cluster or upon 2 nodes of a 8 cores-per-node cluster.

Based on the design architecture described on section 5.3, figure 5.8 provides a graphical representation of the particular requirements of this technique. As we see the particular `D-Cpuset Manager` has to make use of one core per node in order to guarantee the dynamic operations of the CPUSET Mapping technique.

This malleability prototype defines the rigid part (normal job) of a malleable job to occupy one core of each participating node. Initially all the tasks of the application will be bound upon one

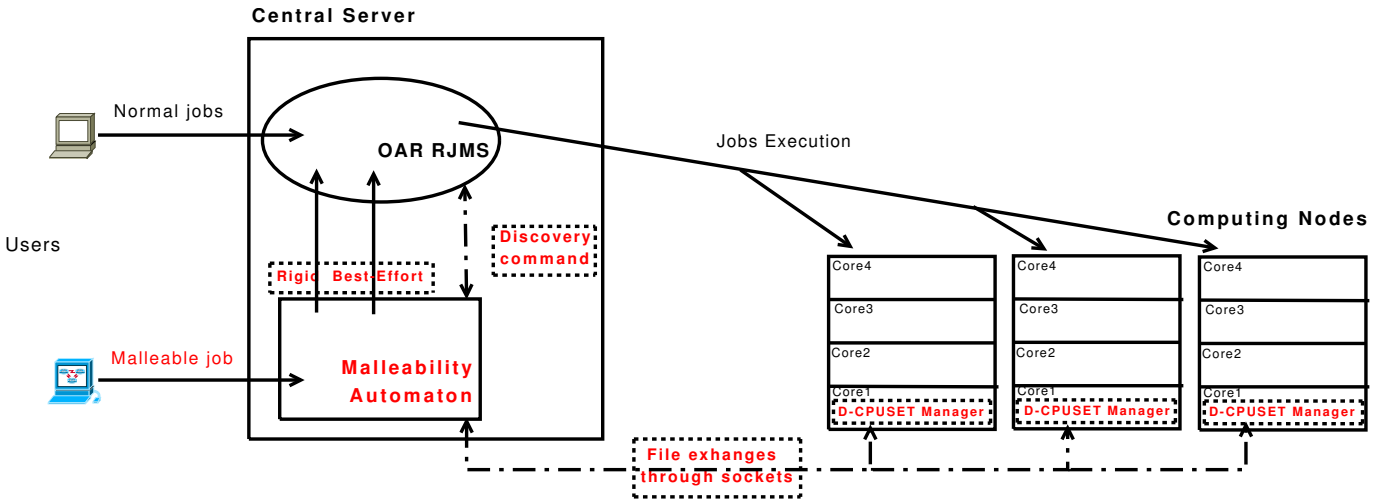


Figure 5.8: Dynamic CPUSETs Mapping architecture

single core per node. When more cores of the same nodes become available, then Best Effort jobs are further submitted and the `malleability automaton` notifies the `D-Cpuset Manager` to perform expanding operations so that the application processes that were sharing a core migrate to the newly available cores. In the opposite case, when an external high priority job asks for resources, some resources need to be relieved and hence Best Effort jobs are killed. In this case, the CPUSET mapping technique performs a folding of processes on the fewer remaining cores, and the demanded cores are given to the arriving higher priority job. Therefore, malleability is achieved by the use of Best Effort jobs and the CPUSET folding and expanding of processes. Figure 5.9 presents some scenarios, showing the different stages for only one of the participating nodes of the cluster.

It seems that besides the restrictions, this system level approach can actually provide malleability without complicating the function of OAR resource manager. Nevertheless, there are issues that have to be taken into account. The overhead of the expanding or folding operation upon the application has to be measured. Furthermore, since our context concerns cluster of shared memory architectures, it will be interesting to see how two different MPI applications running upon different cores on the same node, would perform during the expanding and folding phases.

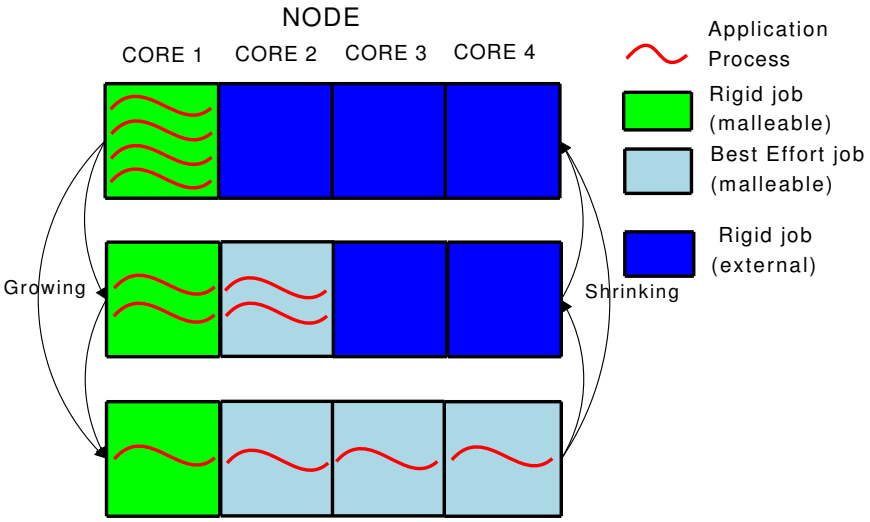


Figure 5.9: Behaviour of CPUSET mapping technique upon one of the participating nodes.

5.4.2 Evaluating Dynamic CPUSETs Mapping technique

In this series of experiments we evaluate the impact of the CPUSETs expansion upon NAS benchmarks. We have particularly chose BT and CG class C NAS benchmarks. These two benchmarks cover a wide spectrum of applications and communication patterns. In Figure 5.10 we present the results of NAS BT application performing static and dynamic CPUSET expansion. In each case the 'Static-1/4cores/node' box implies the use of 1 core per participating node that is the minimum folding of processes that can be applied. On the other hand, 'Static-4/4cores/node' box implies the use of 4 cores per node, which represents the maximum expansion of processes that can be made upon a dualCPU/dualCore architecture. For instance, in the case of BT-36 we use 9 nodes with 4 processes running upon each node: 4 processes on 1 core for the Static-1/4 case and 1 process on 1 core for the Static 4/4 case. The 3 boxes between the above 'Static-1/4' and 'Static-4/4' instances represent different dynamic instances using the dynamic CPUSET mapping approach. All 3 instances imply the use of 1 core per participating node at the beginning of the execution performing a dynamic CPUSET expansion to use 4 cores after specific time intervals. The expansion trigger moment is placed on the 25%, 50% and 75% of the 'Static-1/4cores/node' execution time of each BT.

It is interesting to see the speedup between the Static cases of '1/4' and '4/4' cores/node among each different BT. We observe that this speedup becomes more important as the number of BT

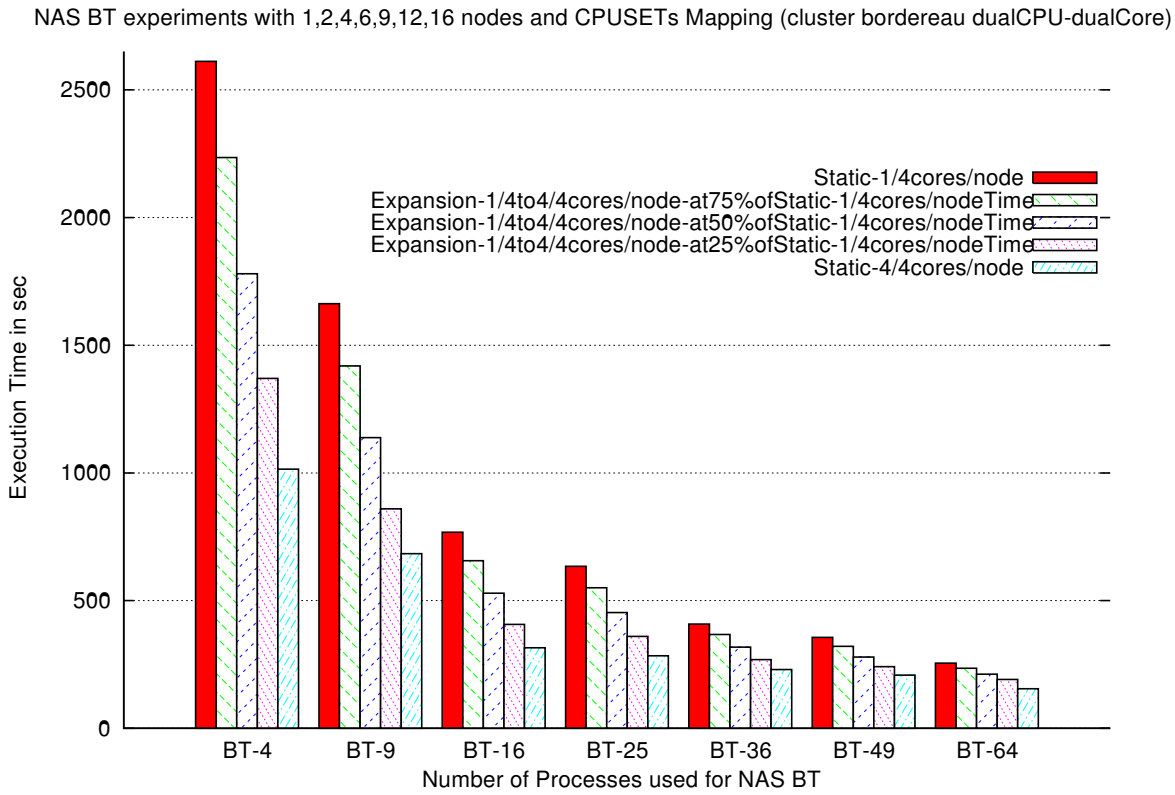


Figure 5.10: NAS BT-(4,9,16,25,36,49,64) behavior with Static and Dynamic CPUSET mapping operations with direct expansion from 1 to 4 cores.

processes go smaller. In the case of BT-4 the speedup from the use of 4/4 cores/node to 1/4 is 1597 sec. which means 61,5% of the 1/4 cores/node time. Whereas the relevant speedup of BT-25 is 350 sec. or 53,8% of the 1/4 time, and to BT-64 is 100 sec. or 33,3% of the 1/4 cores/node time. Furthermore, the figure shows that the technique used for the *dynamic CPUSET mapping* expansion from 1 to 4 cores works fine without complicated behaviour. Linear increase of the expansion trigger moment results in a linear increase of the execution time.

Figure 5.11 illustrates the behaviour of NAS BT-36 execution upon 9 nodes. The figure shows Static and Dynamic CPUSET mapping executions along with direct and gradual expansion cases. Considering the static cases (with no expansions during the execution time), we observe a really marginal speedup between the 2/4 to 3/4 and 3/4 to 4/4 cores/node cases (5 sec. and 25 sec. respectively) in contrast with an important speedup between the 1/4 to 2/4 cases (200 sec.). This could be partially explained by a communication congestion for BT after using more than 2cores/node. The

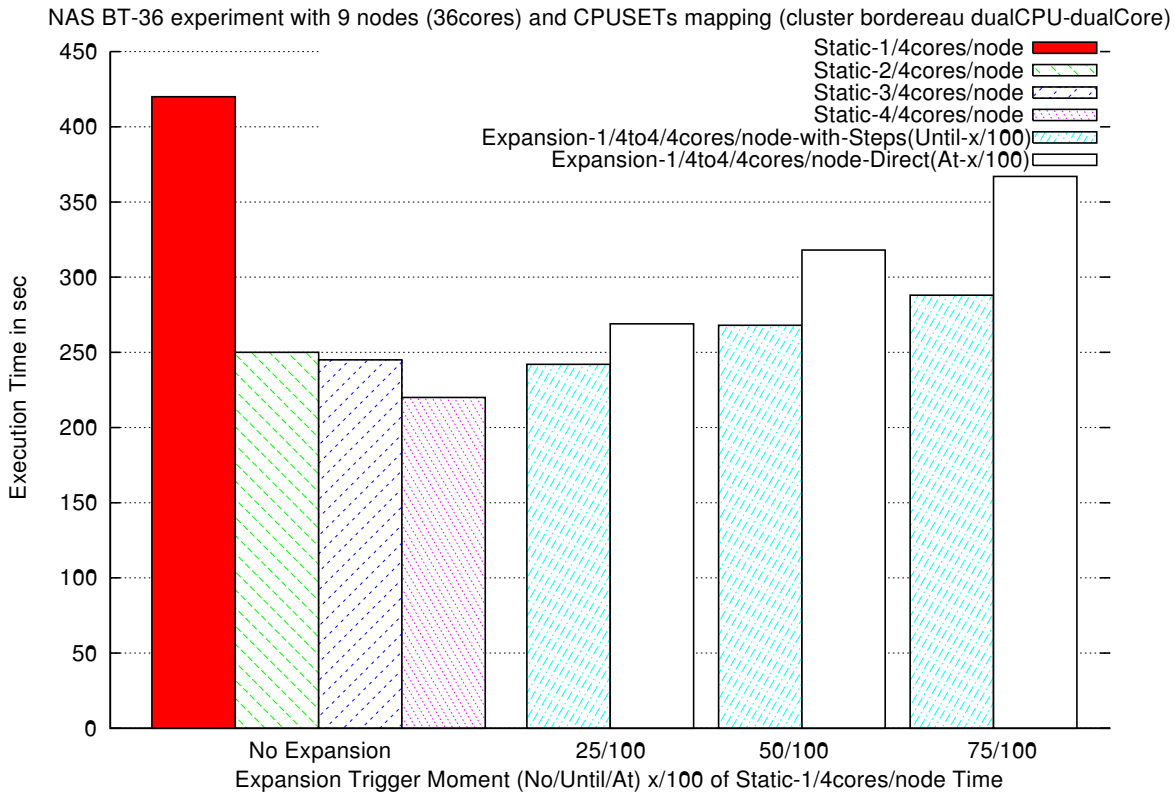


Figure 5.11: NAS BT-36 behavior with Static and Dynamic CPUSET mapping operation with gradual and direct expansion from 1 to 4 cores.

histograms representing the dynamic CPUSET direct and gradual expansions, show us that gradual expansions result in faster execution times. Indeed for each case of the gradual expansion, we perform 3 expansions: 1/4 to 2/4, 2/4 to 3/4 and 3/4 to 4/4 cores/node until a specific instant. On the opposite case of direct expansion we perform only one: from 1/4 to 4/4 cores/node at the above same instant. Thus, after this same instant, both cases are using 4/4 cores/node. The difference is that in the first one there has already taken place 3 expansions whereas in the second one only 1 expansion. For BT benchmark, the results show that it is better to perform gradual small expansions whenever resources are free, in contrast of waiting multiple free resources and perform one bigger direct expansion. Moreover, these results show that the overhead of the expansion process, is marginal, when trading-off with the gain of using the one or more new cores.

Figure 5.12 illustrates the experiments of 3 different cases of BT upon different number of nodes. In more detail we try to measure the impact of the use of different number of nodes - with

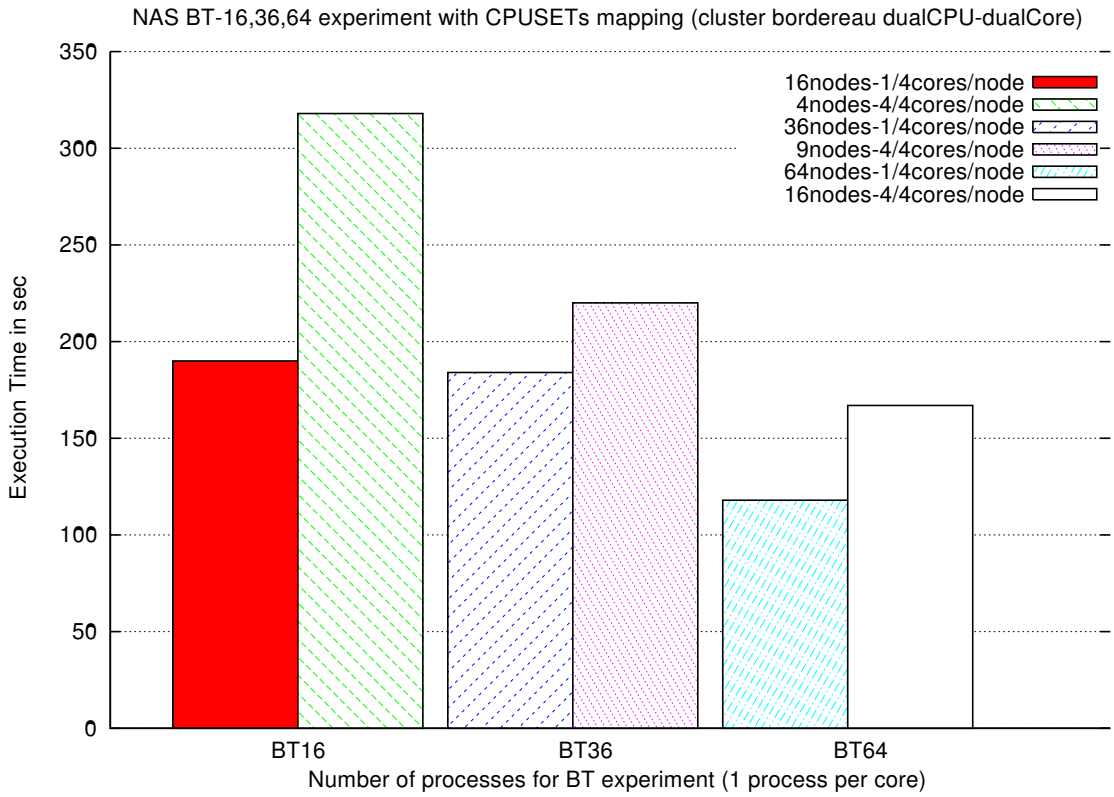


Figure 5.12: NAS (BT-16,36,64) behavior with 1 process per core, different number of nodes and Static CPUSET mapping operations.

one process per core - on their execution time. For every different BT, we experiment with the ideal case of using equal number of nodes with processes (BT-16,36,64 with 16,36 and 64 nodes respectively) along with the case of using equal number of processes with cores (BT-16,36,64 with 4,9 and 16 nodes respectively). It seems that it is always faster to use more nodes. This can be explained by the fact that processes do not share communication links and memory. Nevertheless, this speedup is not very significant implying that, for BT application, the second case is more interesting to use. This is because, we can also perform malleability operations with dynamic CPUSET mapping techniques. Hence, although we have a worst performance we can achieve better overall resources utilization.

Figure 5.13 presents static and dynamic cases in a Dedicated (DM) or Shared Memory (SM) context of BT-64 and CG-64 benchmarks. Concerning the Static and Dedicated memory cases of BT-64 and CG-64 we can observe that the only important speedup is between the use of 1/4 to 2/4 cores/node for BT-64, whereas for all the rest the performance is quite the same. In the

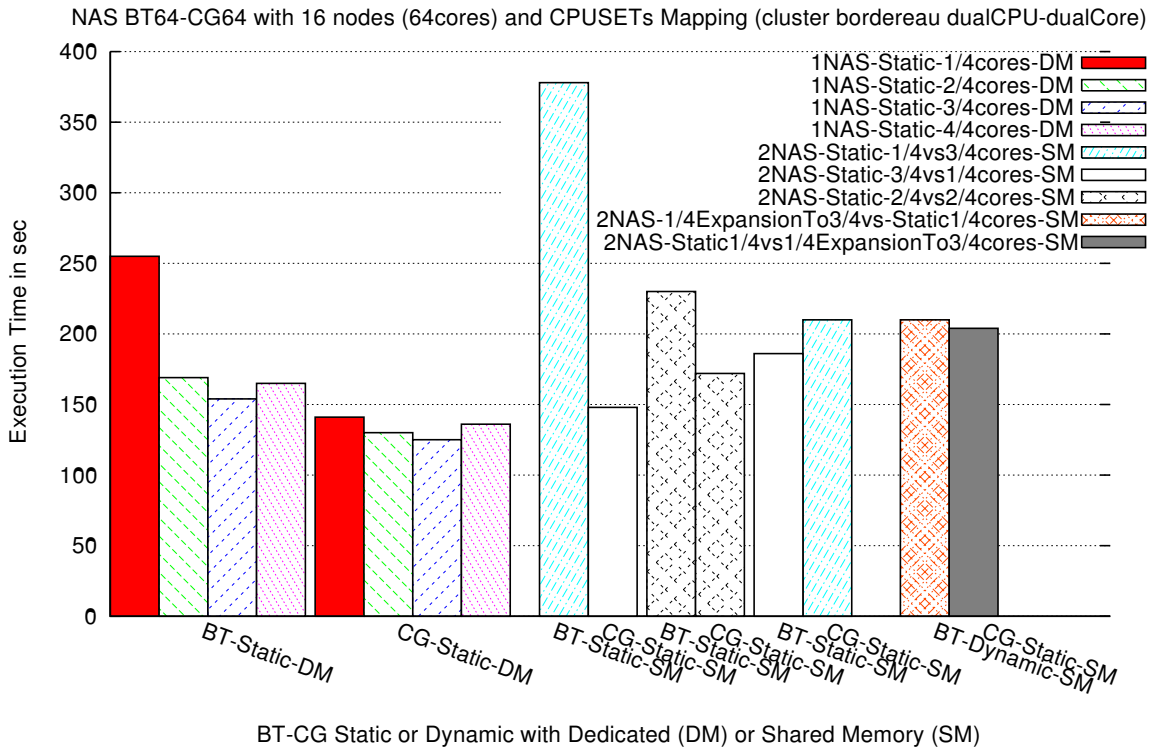


Figure 5.13: NAS BT-64 and CG-64 behavior, with Shared Memory and Static/Dynamic CPUSET mapping operations.

shared memory context, all performances decreased as expected because the benchmarks were sharing memory and communication links. Nevertheless, BT-64 presented an important sensitivity to resources dynamism. This can be observed by the fact that we have a significant performance degradation for BT-64 when it uses 1 core and CG-64 uses 3 cores of the same node; and by the fact that the performance is upgraded for BT-64 when the above roles are inversed. Finally it seems that a dynamic on-the-fly expansion, using from 1 to 3 cores/node, achieves a good performance as compared to the Static cases with shared memory. On the same time the degradation of CG-64 performance is small when BT-64 is expanding.

In the second series of our experiments we will be using BT and CG in a shared memory context. According to those last results, where BT presents more sensitivity in resources dynamism than CG, we decided to implicate BT with malleability whereas CG will represent the rigid applications.

The figures of this section show the results obtained when performing only expanding operations. Nevertheless, our experiments indicated that there is no difference when performing folding operations.

Our experiments validate the dynamic expansion of CPUSETs using specific NAS benchmarks. Nevertheless we argue that all different NAS applications should have been used in order to have a better view of the performance infulences. Intuitevly, We expect that CPU-bound applications will have better speedups than Memory or I/O-bound applications. The choice of BT and CG was made because they make use of similar resources making them good candidates for an average case of applications.

5.5 Improving Resource Utilization using Malleability

The second series of experiments, aims at the evaluation of the two malleability techniques when using automatic submission of real workload traces. The experiments were conducted upon the cluster described on section 5.2.2. For the sake of these experiments 17 nodes of Bordereau cluster were allocated from which 1 node was set as OAR central controller and 16 computing nodes (DualCPU-DualCORE). A workload of 5 hours part of a DAS2 workload (for a cluster of 64 cores) with 40% of cluster utilization is injected into OAR. This workload charges the resources, representing the normal workload of the cluster. At same time one malleable job per time is submitted and will run upon the free resources, i.e. those that are not used by the normal workload.

The main differences of the experiments, for the two different malleability techniques, lay upon the type of application executed and on the way the malleable job is submitted on each case. For the *Dynamic CPUSET mapping* approach we execute BT benchmarks for the malleable job and CG benchmarks for the static workload jobs, so that we can observe the impact on resources utilization in a shared memory context. The malleable job is submitted to OAR in accordance to the guidelines of Section 5.4.1. This means that one core per participating node has to be occupied by the rigid part of the malleable job. Since the time of one NAS BT execution is rather small, especially with big number of processes, we decided that the malleable job will have to execute 8 NAS BT applications in a row. At the same time, CG benchmarks are continuously executed during the normal jobs allocation, as noted in the workload trace file. The number of processes for each NAS execution during the experiment is chosen according to the number of current free resources.

In the experiments of the *Dynamic MPI* case, the malleable job implies the execution of Mandelbrot benchmark. The normal workload jobs (from DAS2 workload traces) are empty 'sleep' jobs, just occupying the resources for a specific time. Since the malleability is performed using whole nodes, there was no need to perform real application executions with the normal workload jobs. As explained on subsection 5.3.1, the malleable job is submitted to OAR, by occupying only one whole node in the rigid part of the malleable job and the `besteffort` part is as large as the amount of remaining resources.

The dynamic executions of both techniques are compared with moldable experiments running the same applications. In more details, the malleable jobs submission is substituted by *moldable-besteffort* jobs submission. As such, we define a *moldable* job that starts its execution upon all free cluster resources and remains without changes until its execution ending. However when some resources are demanded to supply arriving jobs, it will be immediately killed, like a *besteffort* job. Hence a moldable-besteffort job can take advantage of the cluster's otherwise unutilized resources by adapting itself on resources availabilities only before the start of its execution and does not provide any dynamicity after that.

Since the experimental results figures are very similar for both malleability approaches, we include only the figures featuring the support of Dynamic-CPUSET Mapping technique upon OAR. Nevertheless the graphs concerning the Dynamic-MPI technique are provided on the annexes section 8.2. Figures 5.14 and 5.15 show the results of malleable and moldable-besteffort jobs respectively. It is interesting to observe the gain upon the cluster resources utilization to the dynamic context, presented in Figure 5.14, as compared to the moldable case in Figure 5.15. The white vertical lines of Figure 5.14 until 5500 sec., represent the release of `besteffort` jobs resources. This happens, when one group of 8 BT executions are finished until the next malleable job begins and new `besteffort` jobs occupy the free resources. Also, at starting time the malleable job begins, only with the rigid part (i.e only one core) and immediately expands using the other available cores in the `besteffort` part of the malleable job. After 5500 sec. the execution of malleable jobs start to be influenced by jobs from the workload (normal jobs). In that way, the white lines (empty spaces) also mean that the application did not have enough time to grow itself, before a new normal job arrived.

In terms of resources utilisation, since the normal workload makes use of 40% of cluster resources, this leaves a total 60% of free resources. Table 5.3 shows overall results of our executions. Moldable-besteffort jobs use a 32% of the idle resources arriving at 72% of total cluster resources

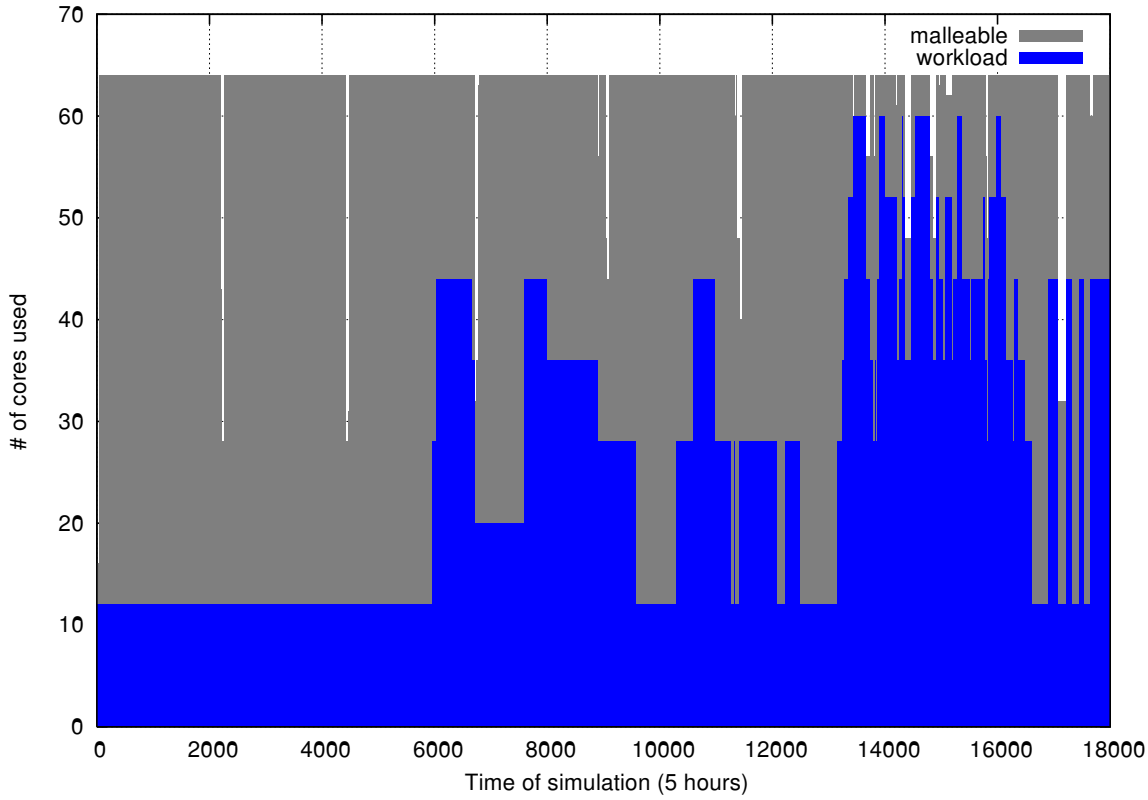


Figure 5.14: Malleable job executing BT application upon the free resources of the normal workload.

Strategies	Overall System Utilization	Terminated Jobs	Error Jobs	Average Waiting Time for rigid jobs
Malleable Dynamic-CPUSET Mapping	97%	8	0	81sec
Malleable Dynamic-MPI	98%	8	0	44sec
Moldable-besteffort	72%	5	4	8 sec

Table 5.3: Comparison of malleable techniques and moldable besteffort during execution of

used. On the other hand, in the dynamic context of malleability approaches, the use of idle resources reach 57% arriving at 97% of overall cluster utilisation. Hence, an improvement of almost **25%** of resources utilization is achieved when the dynamic approaches are compared with the moldable one.

Furthermore, we observed the number of jobs executed during the 5 hours of experimentation. In the dynamic context, we obtained 8 successfully 'Terminated' malleable jobs, compared to 4

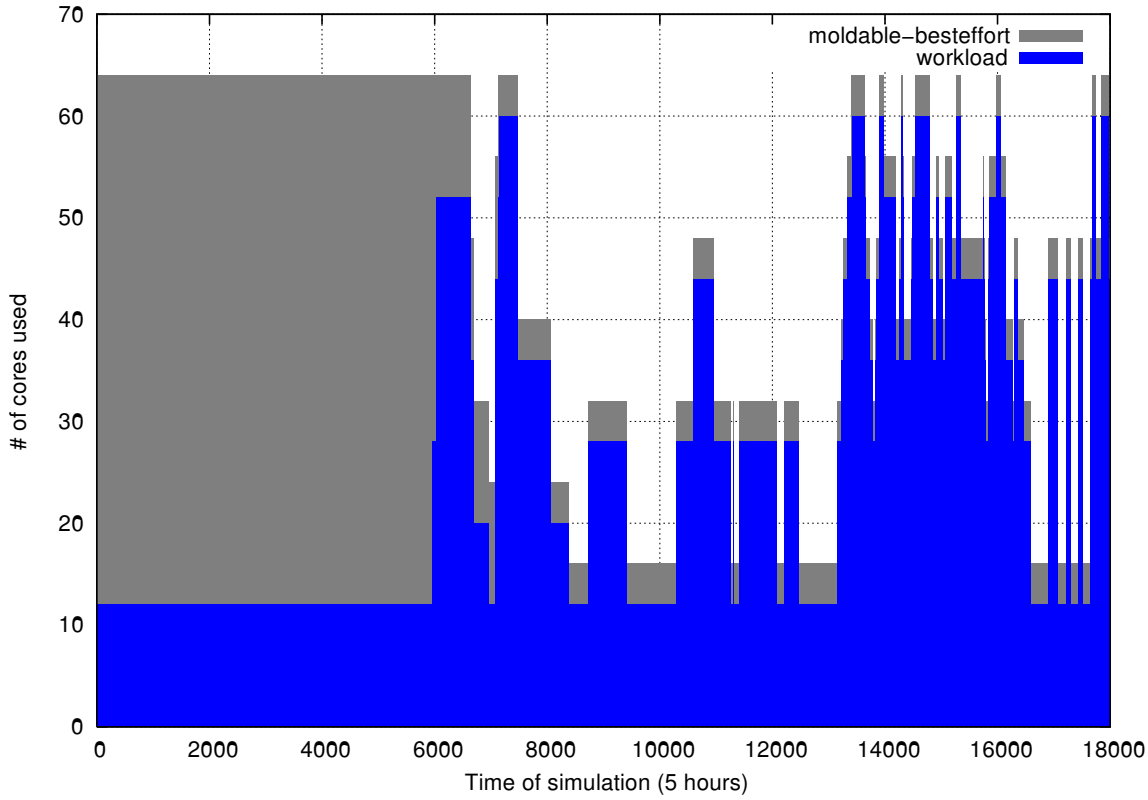


Figure 5.15: Moldable-besteffort job executing BT application upon the free resources of the normal workload.

'Terminated' and 5 in 'Error State', for the moldable one. That means that 5 moldable-besteffort jobs have been started and killed due to need of resources by a rigid job. As expected there were no 'Error State' jobs in the malleable cases, since the rigid part guaranteed succesfull execution. Hence the overall 97% of system's utilization in the *malleable* case represents efficent system utilization. On the other side the overall 72% in the *moldable-besteffort* jobs case does not represent only terminated computations but also not valuable utilization.

Finally the impact of the response time, for the normal workload, was also measured. The results have shown 8 sec. of average response time in the moldable context, compared to 81 sec. of average response time in the dynamic CPUSET mapping technique and 44 sec. in the Dynamic MPI technique. The big response time of the dynamic CPUSET mapping technique, is explained by the allocation of one core per node by the rigid part of the malleable job. This limits the number of free resources for jobs coming from the normal workload. The response time for Dynamic MPI approach, is explained by the 40 sec. grace time delay to the MPI malleable application. This

grace time was added to OAR for the shrinking operations, as presented on Section 5.3.1.

5.6 Conclusions

In this chapter we have designed and implemented prototype techniques upon a resource and job management system, to support malleability on multicore architectures. Two different approaches were proposed that take advantage of the idle periods of clusters normal workloads and allow jobs to adapt themselves according to the varied availabilities. The techniques are implemented as external module of OAR without interfering with the normal functionality of the cluster. They take advantage of the concept of besteffort jobs for the dynamicity and simple communication protocols based on file exchanges between the malleable application and the server.

The malleable Dynamic-MPI technique requires a real malleable parallel application programmed with the `MPI_comm_spawn` primitives to be dynamic according to resources disponibilities but is the least intrusive to the system by keeping only one node as the rigid part of the malleable job. On the other hand, the Dynamic-CPUSET Mapping technique allows to every kind of parallel application to have malleable behaviour through the dynamicity of CPUSET mechanim upon multicore architectures. However this technique is more intrusive to the system since it absolutely needs 1 core per node as the rigid part of the malleable job.

Both techniques attain very high results of optimization of system utilization achieving an improvement of 25% when compared to a moldable-besteffort approach. However this gain comes with a trade-off upon the waiting times of rigid jobs of the normal workload. As expected the loss was bigger in the Dynamic-CPUSET technique. Ofcourse we argue that in a context of pure besteffort jobs like in an environment similar with the lightwight grid Cigri discussed on chapter 3 the overall system utilization would be nearly 100% but the real efficient utilization is much less than that as we have seen in the cas of *moldable-besteffort* jobs and in our experiments on section 3.4.2. At least in the case of our malleability techniques the overall systems' utilizations depict real efficient computations.

Hence, the malleable jobs as prototyped upon OAR system, represent an upgraded version of the CIGRI lightweight grid jobs provided on chapter 4, since they offer the guarantee of execution of rigid jobs followed by the dynamicity of a besteffort job.

Nevertheless, our study has let some aspects in the side. Our initial design considers only one malleable job at a time, for simplicity and better measurement of the side-effects, but future studies are planned for the support of multiple malleable jobs submission. Our experimentation could be

also further extended to take into account more complex applications in both cases of malleability techniques. Concerning the Dynamic-MPI technique we are willing to test a real application reprogrammed into malleable one with the `MPI_comm_spawn` primitives. On the same time the variations of all the NAS benchmarks with different application profiles could provide more valuable results for the shared memory contexts of the Dynamic-CPUSET Mapping technique, where we believe that more research is needed. The cases of memory swaping upon the disk has not been taken into account in our study. Concerning the evaluation more trace files with various system utilizations should be used. In the same time the workload profiles could be taken into account because it's obvious that simple workloads with not a lot of variations and a small number of big jobs could benefit more the *moldable-besteffort* approach than the *malleable* ones.

Chapter 6

Energy Efficient Management Techniques

Energy efficiency has become an important research domain in all areas of science. In Large-Scale Distributed Systems and High Performance Computing, the energy consumption plays a significant role in the evolution of these systems. The increase in computation performance has come with an even greater increase in energy consumption. The last few years High Performance Computing Systems has started to be limited by factors like power usage, heat dissipation and the resulting bills for power and cooling.

Research efforts upon all different abstraction layers of computer science, from hardware upto applications, strive to improve energy conservations. As far as the systems middleware concerns the Resource and Job Management System (RJMS) can play an important role in this game since it has both knowledge of the hardware components along with information upon the users workloads and the executed applications. In previous chapters we have studied methods to take advantage of otherwise idle resources in order to achieve better system utilization. According to prior work, idle computing resources consume a very important amount of energy [19],[20]. This energy could be gained if specific actions could take place. Hence in this chapter, we extend our research in order to take advantage of otherwise idle resources so as to achieve energy-efficient system exploitation.

In particular we have implemented a resource management extension upon a versatile resource and job management system so as to power-off otherwise idle computing machines of a cluster under specific conditions, depending on the users workloads. Following this method a cluster can benefit of its idle periods and perform energy reductions. Conversely, if a job demands powered-off machines the RJMS boots them and allocates them when they are powered-on. Nevertheless, the impact upon the jobs response time and applications performance should also be taken into account as trading-off the energy reductions.

In an effort to take full advantage of the privileged position of the RJMS we extended the green resource management optimizations so as to motor energy awareness on the user level. Hence, we have implemented mechanisms upon the RJMS to help users achieve energy reductions on the application layer.

Indeed a lot of efforts have been made on this layer to provide energy-efficient MPI programming [21],[197] or special Dynamic Voltage and frequency scaling mechanisms for reducing energy on MPI programs [22],[198]. In order to encourage this kind of research on the application layer we developed specific options upon the RJMS. In more detail, we have developed a particular type of jobs that can take advantage of adapted CPU voltage/frequency scaling and Hard disk spin-down techniques upon modern platforms that provide this kind of hardware treatment. Therefore, users are enabled to perform optimizations upon their applications programming or executions, by having energy consumption in mind. Our study evaluates the trade-offs between energy consumption and performance for various MPI applications.

6.1 Related Work

The increasing demands for energy on High Performance Computing the last few years have started to provide an important issue to the evolution of these systems. Hence, a new axe of research upon energy reduction on HPC emerged as an imminent neccesity. A way to treat this growing problem is to improve the energy efficiency at different layers of abstraction. Research upon hardware [199], [200] and multiprocessor technologies [201, 202] result into less-energy demanding components. Improvements on microchips architectures optimize the trade-offs between energy and performance [203]. The evolution on the hardware level helped for the construction of new low-power High Performance Computing Systems. Systems like Green Destiny [204] addressed these problems by significantly reducing per-node power consumption.

While TOP500 list [3] maintains the 500 most powerfull computer systems in the world, a new complementary list has been created called Green500 [18] that presents the 500 most energy-efficient systems. This list has been created in an effort to provide focus not only in performance (as measured in floating-point operations per second: FLOPS) and targeted by TOP500 list, but on other metrics like performance per energy consumed (measured in FLOPS/Watt).

Important research has been made on the application layer in order to analyze and control the energy efficiency of MPI programs [21],[197] or provide special Dynamic Voltage and frequency scaling mechanisms for reducing energy upon MPI applications [22],[198]. Furthermore users are

starting to become more energy conscious and in the same time they know better the hardware needs of their applications. There are programs that use more the CPU and the network without disk I/O while others perform a lot of disk access and no message passing.

One common mechanism on newer microprocessors is the mechanism of on the fly frequency and voltage selection called DVFS. Reducing CPU frequency leads to reducing energy consumption. As observed on previous work adaptive use of frequency scaling may result in important energy benefits for small performance losses [205]. A similar technique for reducing energy consumption exists for hard disks. Disks are made to service requests at their maximum speeds. Even if a disk is not servicing any requests, it continues to spin at the maximum rotational speed and hence wastes energy. Therefore the possibility of spinning down the disk when not in use may contribute in energy efficiency [206]. Our implementations for energy-reductions during jobs execution are based on these ideas and are presented in detail in section 6.4.

Concerning related work upon energy efficient Resource and Job Management Systems, Moab [1], PBSPro, LSF, Condor, SGE and Slurm [2] [102] advertise the support of adapted energy efficient techniques, based on exploitation of idle resources. The basic ideas for energy-reductions through the technique of machine power ON/OFF during unutilization periods, are the same upon all systems. Nevertheless, as far as our knowledge, there is no published work with evaluation of their mechanisms. Moreover, no similar mechanisms have been found on other RJMS that allow the exploitation of hardware DVFS techniques for individual job energy-conservations during execution. Only SLURM software offered a solution for CPUFREQ exploitation for energy-reductions which was parametrized only by the administrator. In this chapter we present also experimentations with SLURM Energy conservation, automatic power ON/OFF, mechanisms and we evaluated and compared its performance with our implemented Green Management techniques upon OAR system.

Other research teams have been focused around the subject of dynamic thermal management of data centers. Researchers on HP-labs have been studying and developing real-world scheduling algorithms that take into account the real-time thermodynamic formulation of a data-center room by spotting the hot and cold spots [207, 208]. Based on these information they explore temperature-aware workload placement algorithms that can lead to lower temperatures and higher energy consumptions in the data centers.

All our developments for energy-efficient computing took place upon OAR resource and job management system. The research presented in this chapter is a part of an integrated framework

[1] $http://www.clusterresources.com/solutions/green-computing.php$
[2] $https://computing.llnl.gov/linux/slurm/power_save.html$

destined for energy conservations for High Performance Computing called Green-Net [209], [210] [3], which is a research collaboration between 3 laboratories.

6.2 Measuring Energy Consumption upon Grid5000

In the context of energy saving in HPC, users should be aware of what exactly their applications consume. This requirement has led to developing new information systems for end users. For instance, in a similar context Google with its PowerMeter [211] and Microsoft with its Hohm project [212] will allow users to view precisely their energy consumption at home and to receive advices on how to reduce their every days energy consumption. One key point of these two systems is that they aggregate the data about energy usage over several months and even years, helping the users to understand how their consumption evolves over time. Similar approaches can be taken in the HPC systems contexts.

The *Green-Net* [209], [210] framework proposes a top-down approach on three levels. The first level represents the information delivered to the users to raise their energy consumption awareness. The second involves the users in trading performances to energy consumption. The last level represents the automatic behavior and adaptation of the distributed system to gain energy. The context of Green-Net 6.1 is the large scale distributed systems, more precisely grids and clouds, where a local resource and job management system handles the resources and distributes them among the users jobs.

In this environment our research is focused on the RJMS level and we use the interfacing part of Green-Net to the energy consumption measuring capabilities of Gri5000 platform, in order to effectuate the performance evaluations of our implementations.

6.2.1 The Hardware Energy-Meters

Current technology usually does not allow us to measure how much energy consume the individual components of a computer. The only possibility is to measure the consumption of a whole computer. Our experiments currently use two types of measurement appliances: the Hameg and the Omegawatt box.

The Hameg is an electronic laboratory power meter (HAMEG HM8115-2) [4], which offers a

[3]This research is supported by the GREEN-NET INRIA Cooperative Research Action: http://www.ens-lyon.fr/LIP/RESO/Projects/GREEN-NET/

[4]http://www.hameg.com/147.0.html

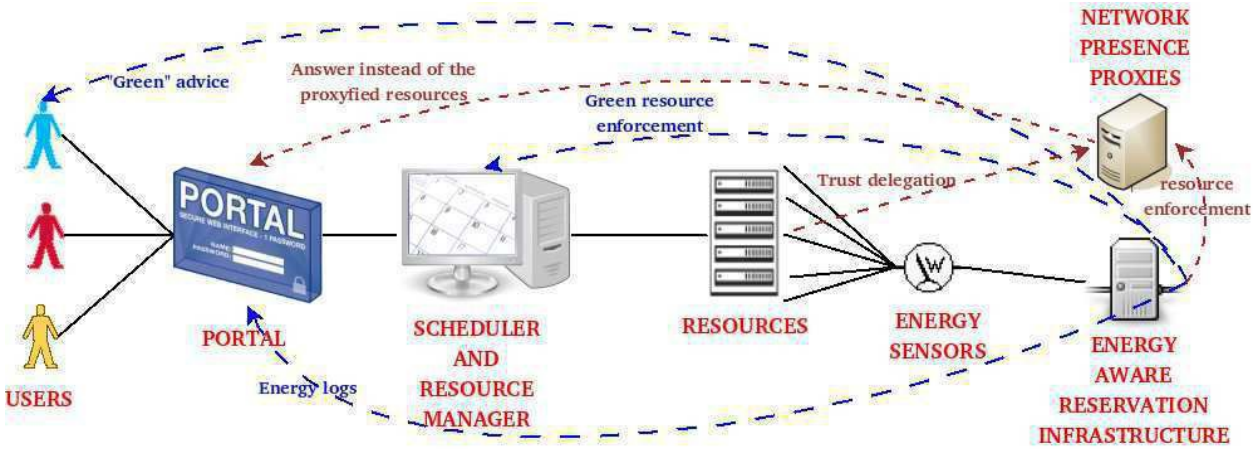


Figure 6.1: The GREEN-NET framework.

precision of 0.5%. It can measure the energy consumed by only one element (usually a computer), but can provide all the relevant information (effective and reactive power, phase, etc). As this power meter is calibrated for laboratory use, it is utilized for verification of the other systems.

The Omegawatt box [5] is a customized box produced by Omegawatt to measure the consumption of several nodes at the same time. The current system monitors 162 nodes and is deployed on three sites of Grid'5000, located at Grenoble, Lyon, and Toulouse. We use 6-port boxes in Grenoble and Toulouse, and 48-port boxes in Lyon. These systems are able to take a measurement per second. Each site contains a server responsible for logging the measurements using a dedicated library to capture and handle the data.

The interface with Hameg and Omegawatt equipments uses serial ports, but scientists of Green-Net project are currently investigating alternative hardware (i.e. Plogg) that communicate via Bluetooth.

6.2.2 A Library to Interface with Energy-Meters

Obtaining energy consumption information from several heterogeneous sensors such as those described above is a challenging task. Green-net projects researchers have chosen to develop a library that simplifies the measurements. As requirements this library had to be optimized for near "real time" data acquisition and be extensible to take into account future developments; hence, its implementation has been done using the object-oriented language C++.

[5] http://www.omegawatt.fr/gb/2_materiel.html

Having completed the software layer to interface with sensors, a particular development had to be made to implement client-side applications that collect and log energy comsumption measurements from sensors at time intervals; and applications that access previously collected information. Each measurement contains a time-stamp that specifies when it was performed. On a server responsible for controlling a number of energy sensors, a thread is associated with each sensor to capture the energy consumption of the monitored devices at each second. An additional thread is responsible for responding to requests made by client applications. Clients-server communications are done via Remote Procedure Call (RPC).

- A Web page (Figure 6.2) that provides the energy consumption graphs over time for a number of monitored nodes. Using rrd-tools it shows at different time scales, from minutes to months, the energy consumed by nodes without any context. It is mainly used to give a fast and simple feedback to the user, who has to select only the nodes relevant to her application.
- A Web service that using XML provides the energy consumption data from nodes, based on a list of nodes and a time frame given by the user. Hence, the user can obtain exactly the relevant energy consumption data concerning a job that has run on the grid.

An example of energy monitoring through Green-Net interfaces upon Grid5000 platform is shown in figure 6.2

6.3 Energy Reduction through idle resources manipulation

In this section we analyze the adaptation of a Resource and Job Management System with Energy Efficient features.

According to prior work, idle computing resources consume an important amount of energy [19],[20]. This energy could be gained if specific actions could take place while the machines are not allocated by a user. Figure 6.3 shows a plot of Watt consumption for a single machine when idle, powered-OFF and powered-ON again. It is interesting to observe the 60% difference on Watt consumption between powered-OFF and idle powered-ON states. Intuitively, we can imagine that the gain in energy would be large when we consider a big number of idle cluster computing nodes. Nevertheless, the instant peaks of energy consumption observed in the end of the power-ON phase; make us understand that the time a machine needs to stay in Power-OFF state has to be considerably large so as to result in a gain of energy rather in loss. Another issue that needs to be taken into account is the time that a particular machine needs in order to perform a complete power-on from

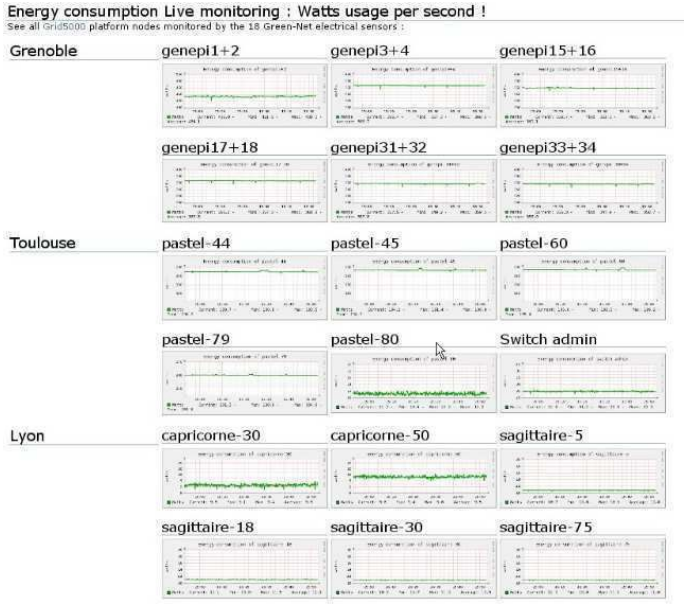


Figure 6.2: Web page example of energy monitoring of 18 nodes.

the instant that the command is issued until the moment that the machine is ready to start executing a task. This can definitely influence the response time of a particular job which is waiting for a node to wake-up to perform computations. Table6.1 shows the duration of the shutdown and poweron phases for a particular cluster in Grid5000. These values change depending the architecture.

Cluster/PowerOFF-ON	Shutdown Time	Reboot Time
Capricorne	15sec	154sec

Table 6.1: Time for PowerOFF-ON

The privileged position of the resource and job management system, which collects information for both resources availabilities and users workloads makes it an ideal tool for integrating actions to be triggered upon the idle cluster resources for efficient energy consumption.

6.3.1 Adapting OAR to exploit idle resources for energy conservations

Under this context we have extended the resource management mechanisms of OAR [59] in order to achieve energy-efficient system exploitation by manipulating idle resources. In particular we

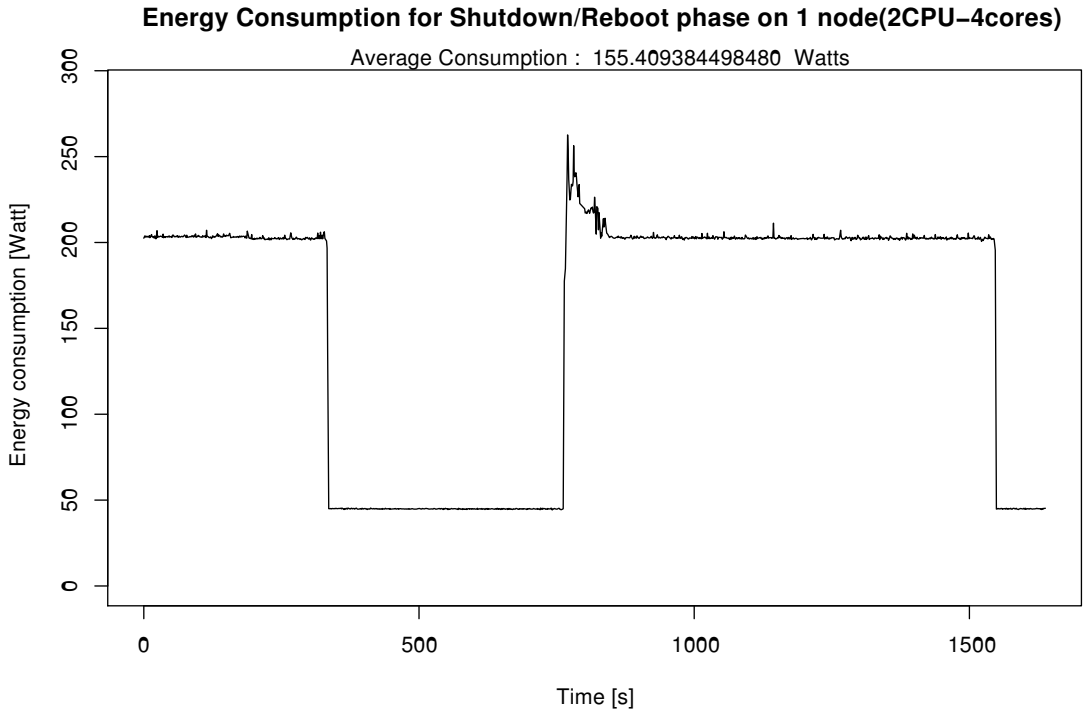
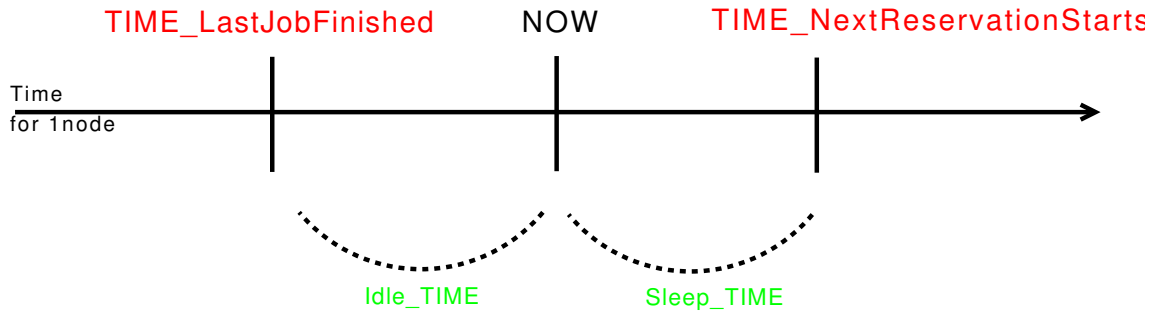


Figure 6.3: One Node Energy consumption Reboot phase

have implemented a resource management optimization upon OAR that powers-off idle computing machines of a cluster under specific conditions. In contrast with the exploitation of idle resources, for more efficient computations, as visioned on previous chapters; In this study our goal is to use this resource idleness in order to achieve energy-efficiency. Following this method a cluster can benefit of its idle periods and perform energy reductions. Conversely, if a job demands machines which are in power-off states, OAR triggers a power-on command and allocates them for the job when they are on Alive state.

Figure 6.4 describes the simple algorithm of the Green Management mode. It uses specific variables that need to be parameterized by the administrator of the cluster. The `Idle_TIME` represents the duration that a machine has not been allocated by a job. Similarly the `Sleep_TIME` represents the duration that the machine will remain unallocated, which means that a job reservation is not planned upon it. As shown on the figure, the Green Management algorithm examines the idleness conditions, according to the parameterized variables, and executes the predefined actions upon the machines. The executed commands depend on the platform and operating system but could be any

type of shutdown, standby or hibernate modes along with the relevant wake-up commands. In case of sudden job arrival which needs the machine, the wake-up command should power-on the machine and OAR initiates the job when all machines are ready for utilization. Ofcourse a significant jobs waiting-time is expected depending on the reboot time of the machines.



Algorithm for OAR Green Management

```
Node GoToSleep
        if    $Idle_TIME > A_PreDefined_Idle_TIME
                AND
              $Sleep_TIME > A_PreDefined_Sleep_TIME
        then
                  exec GoToSleep_Command

Node WakeUp
        if    SleepingNode_isNeeded  then
                  exec WakeUp_Command
```

Figure 6.4: Green Management mode upon OAR

The implementation of this mechanism upon OAR was rather straightforward with the definition of the logic upon the MetaScheduler module which calls the scheduling module as described on chapter 3. The MetaScheduler makes the necessary checks to see if resources fullfill the conditions to GoToSleep or if some need to be Waken-UP. The MetaScheduler takes the decisions and another module is responsible for issuing the relevant commands. The commands need to include specific Timeouts in order to guarantee that if a resource is not Alive after a particular time then it has to be suspended and not used.

Newer versions of the OAR Green Management technique consider additional optimization techniques of the initial implementation.In order to optimize response times for small or interactive jobs that need small amount of resources some resources are kept alive during the process. Power-off and power-on with groups of nodes in order to avoid sudden electricity variations. However

those features, which are already in production, were not finished at the time of the experimentation so they have not been evaluated.

Another study that we conducted was to enhance the above algorithm with a predictor based upon historical data. This was motivated by the fact that the load is likely to be more during the week days than during nights or during the weekends. Hence we wanted to provide a model that can anticipate which days and hours of the week will be idle periods in the cluster and use this information for future reference in order to power-off nodes during periods with low workload traffic and wake them up when the module predicts that the traffic will become larger. The prediction model is based upon a past repository, which aids in maintaining the periodic load of the system and an algorithm which scans for current and future workload and tries to correlate with the past load. history. Even if the study resulted into an actual implementation by Kamal Sharma, the particular prototype was never experimented or used in production [209].

6.3.2 Performance Evaluation of OAR Green Management technique

In order to evaluate our implementation for automatic energy reduction through OAR Green management we have executed experiments upon Grid5000 platform. For this, we have used workload traces from the DAS2 [213] clusters. In particular we have extracted specific parts of the traces according to the system utilization percentage and we have replayed those traces using OAR upon a cluster of the same size deployed upon Grid5000. Our goal is to evaluate the different management modes (Normal and Green) of OAR using different workloads. In this first series of experiments our workloads consist of simple sleep jobs.

Our experiments were made upon Grid5000 on Lyon site and Capricorne cluster with AMD Opteron 246 Dual CPU (2.0GHz/1MB/400MHz), 2GB memory and Gigabit Ethernet network. Since the trace file was collected by a 32nodes(DualCPU) cluster, we have selected 33 nodes of Lyon Capricorne cluster and deployed OAR frontal server upon the one of them and 32 OAR computing nodes. In this first series of experiments our workloads consist of simple sleep jobs since we are only interested to evaluate the system behaviour when idle nodes shutdown or not during the experiments. Our goal is to observe the differences on energy consumption and jobs waiting-time for the normal scheduling mode where idle machines remain powered-ON, compared to the green scheduling mode where idle machines are notified to power-OFF if the particular conditions are fullfilled.

For our tests we have used workload traces of 50.32% and 89.62% system utilization. Each figure, 6.5 and 6.6 present two different experiment plots one with normal and one with green

management upon the same machines. It is interesting to see that the energy gain in both workload cases is very important.

We can note that between the two figures there are differences in energy consumption of idle state (difference of 2KW.h in Total consumption of Normal scheduling mode), even if we are using the same cluster. This difference is due to the fact that we didn't reserve exactly the same machines for the experiments shown on the different figures. It is a fact that there is a significant difference in instant energy consumption between the nodes of this same cluster. Indeed, in the particular cluster it has been observed that nodes situated on the bottom consume less than those on higher placing upon the shelves.

Parameters/Experiments	Experiment 1	Experiment 2	Experiment 3	Experiment 4
Management Mode	Normal	Green	Normal	Green
System Utilization Percentage	50.32%	50.32%	89.62%	89.62%
Total Number of Jobs	309	309	188	188
Total Energy Consumed	42.7 KW.h	30.6 KW.h	40.7 KW.h	36.6 KW.h
Average Job Waiting time	8 sec	829 sec	1 sec	218 sec

Table 6.2: Total Energy consumption and Jobs Waiting Time for normal and green modes

The significant peaks of energy consumption in the beginning of green scheduling mode of figure 6.6 are due to the fact that machines were just rebooted when the execution has started, whereas it was not the case in all other experiments of both figures. This event was irrelevant with our experimentation and it influences slightly the total energy consumption.

Table 6.2 shows significant results of the experiment runs. As expected the gain on energy consumption is followed by a loss on jobs waiting time. After observation of the workload file we observe that the workload of 89.62% is mostly composed by small jobs whereas the 50.32% workload is composed by a small number of large jobs. This explains the big difference we observe in the waiting times. Hence we observe a 28% gain in energy consumption for a waiting time of about 14 minutes in average or a 10% gain for a waiting time of less than 4 minutes in average. Furthermore, the waiting time includes the machines boot time which has been measured as 154sec, in average, upon the specific machines. The study needs to be continued for other workloads and platforms but the first results show a good trade-off between the gain on overall energy consumption and the loss on jobs waiting time.

In a second part of our experiments with OAR Green and Normal modes we have used a

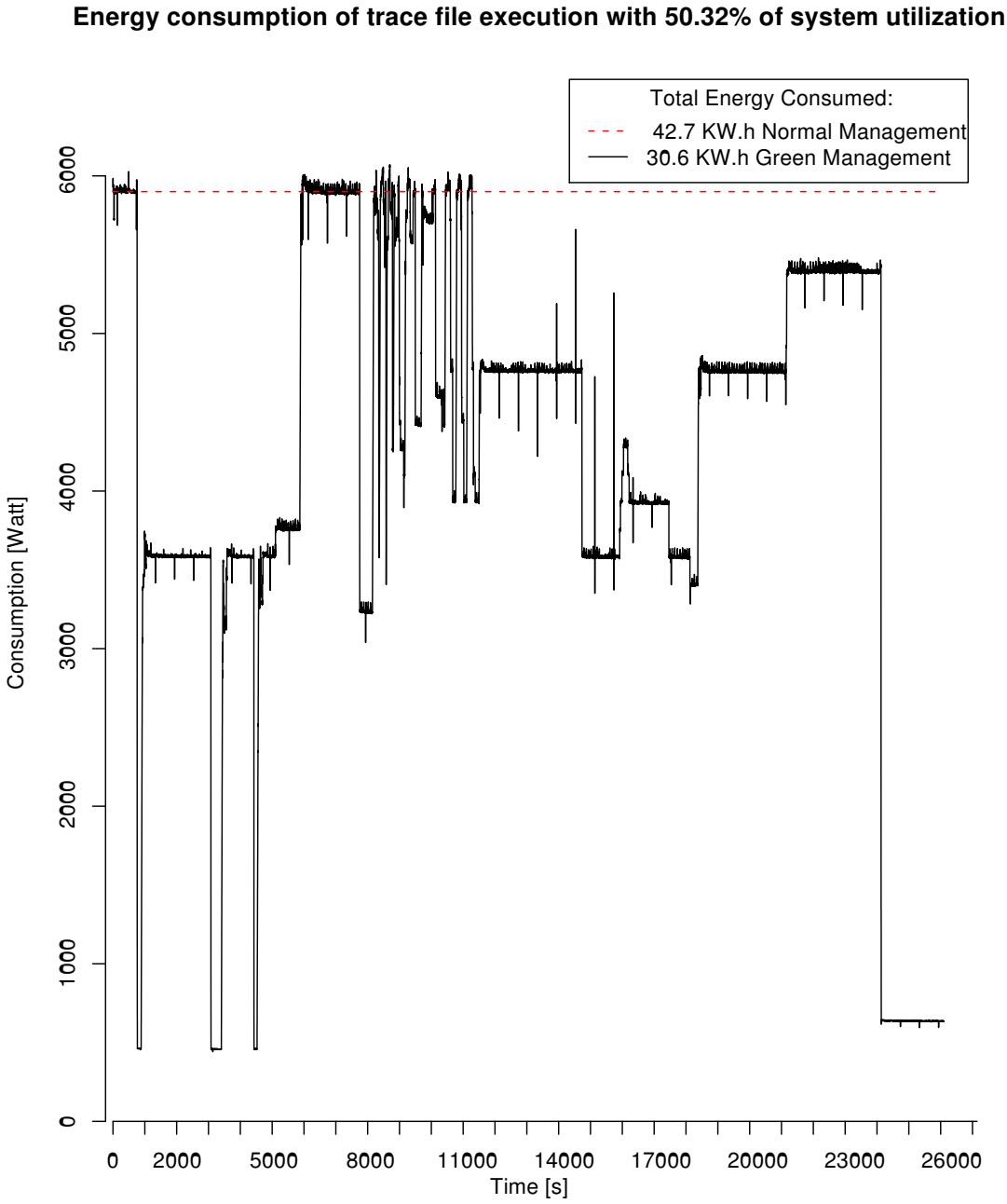


Figure 6.5: Energy Consumption for normal and green management with 50.32% system utilization

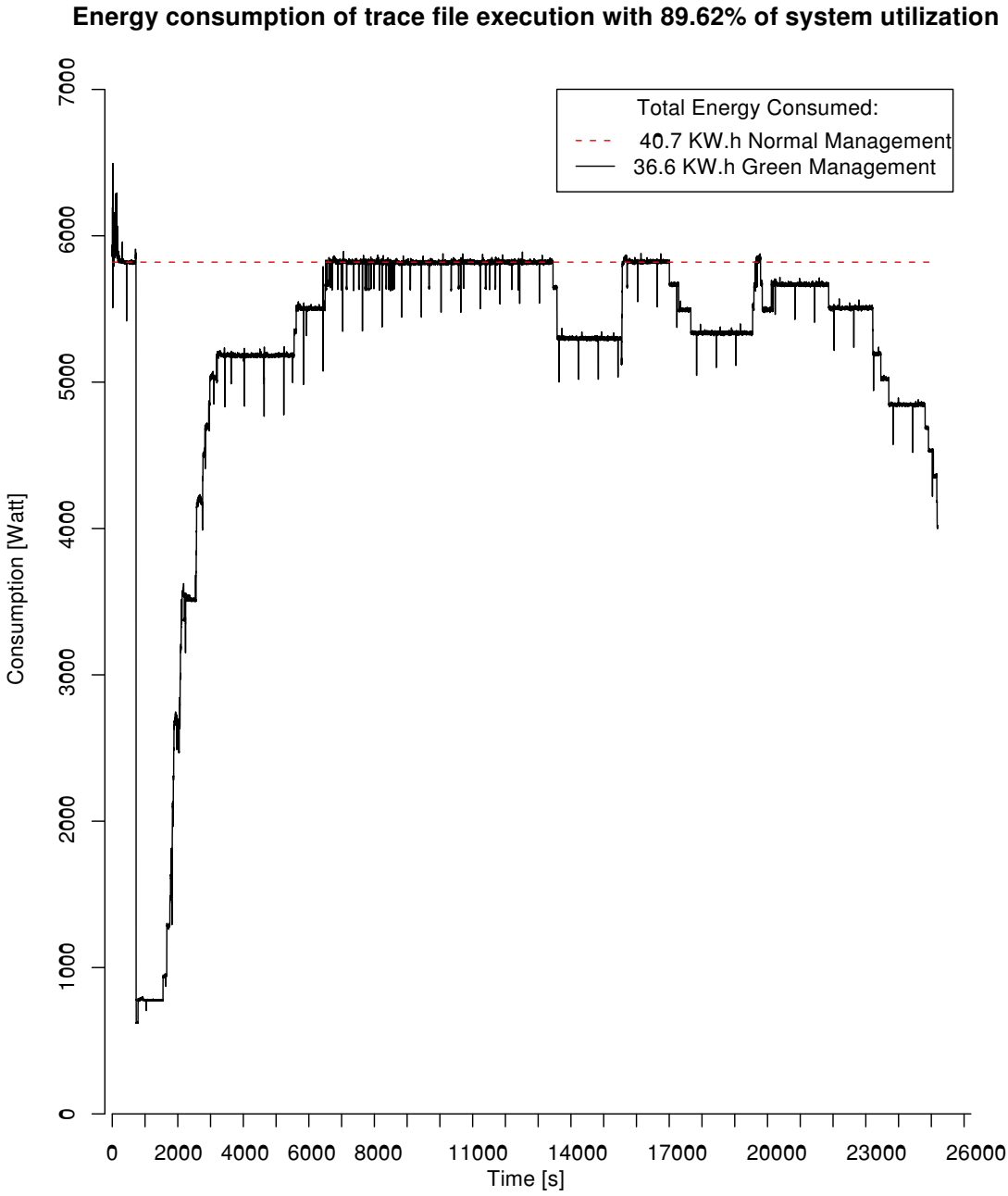


Figure 6.6: Energy Consumption for normal and green management with 89.62% system utilization

different cluster in the same site (Lyon) of Grid5000 platform. The cluster was Sagittaire and had similar characteristics with the previous one (AMD Opteron 250 with 2.4GHz / 1MB / 400MHz ,2 GB of memory and Gigabit Ethernet Network). We have used the same workload for the 89.62% of system utilization but this time the workloads perform real computations of NAS benchmarks. In particular we have utilized BT application of NAS benchmarks and we have implemented a while loop so as to execute BT benchmarks throughout the time that a single job has allocated resources. This will allow us to see the variations of energy consumption with real computations taking place upon the platform. The graph in figure 6.7 shows us the instant energy consumed for each mode when submitting the same workload to the cluster.

It is interesting to observe that the total difference in energy comparison after 5hours of computation is about 7.2KW which gives a gain of 13% of the total computation. Nevertheless we need to observe also the side-effects of this gain in energy.

Figure 6.8 shows the impact on jobs waiting times. More specifically, this impact is provided by a graph of Cumulated Distribution function on jobs Wait time. We can observe that on Normal mode there is almost no wait at all for allocating the resources whereas on Green mode we have a variation on waiting times from 2-3sec up to 400sec with most of the jobs being in an average of about 200sec which is less than 4minutes.

This is an acceptable result, if we consider the gain of 13% of energy consumption for 200sec of waiting time loss. We can also observe the difference on wait-times with the same workloads and different platforms of the previous experiments. The explanation lies on the fact that Sagittarie is a slightly newer cluster than Capricorne and it provides a faster power-on mechanism which makes faster the response of nodes on the demand of OAR to power-on.

6.3.3 Comparison of OAR and SLURM Green Management techniques

In this suite of experiments we compare the Green Management technique of OAR along with that of SLURM resource manager. Both of them provide the same capabilities for performing energy reductions. The comparison was made with earlier versions where they did not yet support the power-on/off with groups of nodes.

The experimental methodology and the platform are the same as in the second part of the previous experiments. Thus the deployment takes place upon Sagittaire cluster (AMD Opteron 250 with 2.4GHz / 1MB / 400MHz ,2 GB of memory and Gigabit Ethernet Network) and the same workload for the 89.62% of system utilization with real computations of NAS BT benchmarks.

Both Green modes are presented on figure 6.9 where we can observe that the final energy

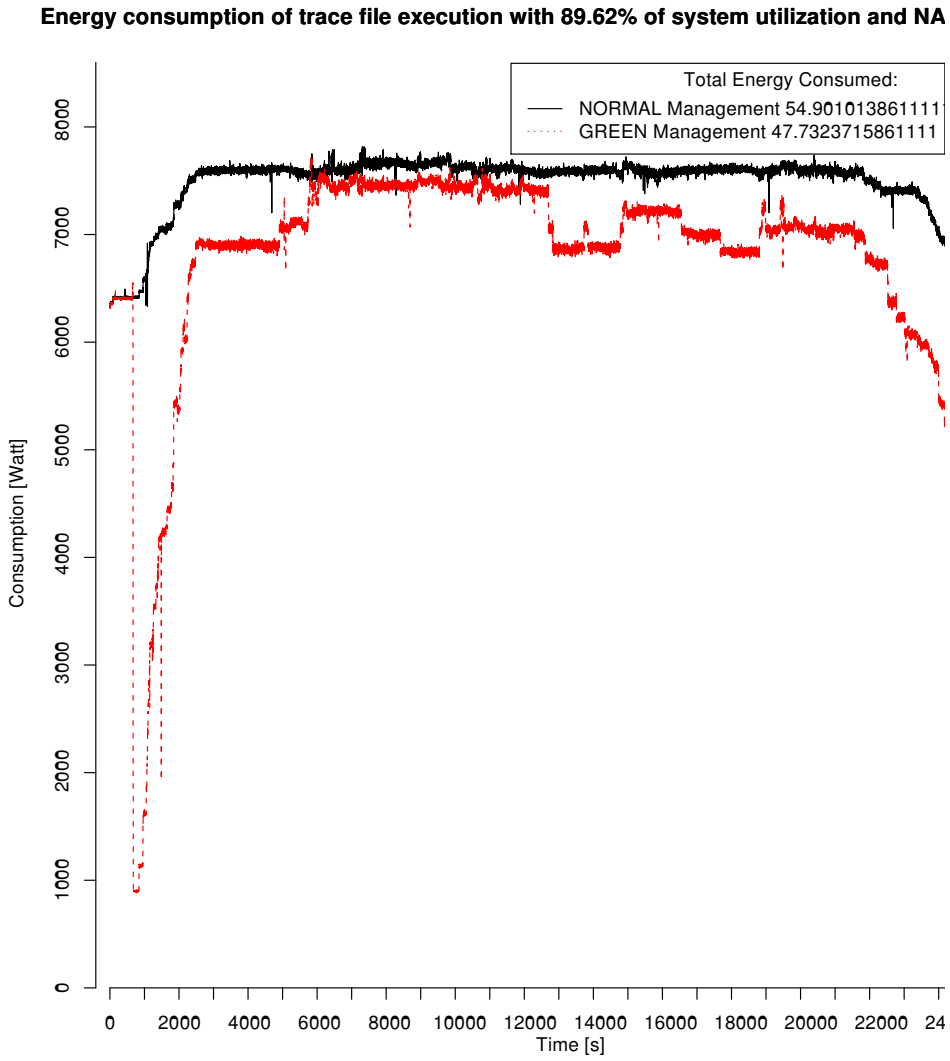


Figure 6.7: Energy Consumption with trace file of 89.62% of system utilization and NAS BT Benchmark upon a 32 nodes(biCPU) cluster with OAR

consumption after 5hours of experimentation is the same even if the instant energy consumption varies between the two cases. The variations between the two graphs representing the instant energy consumption can be explained by slight differences in scheduling and tasks placement decisions. Even if both RJMS have been configured to use the same scheduling policy which is conservative backfilling and both used task placement techniques with fine granularities; the packing of the jobs and the internal matching decisions can be different from one system to the

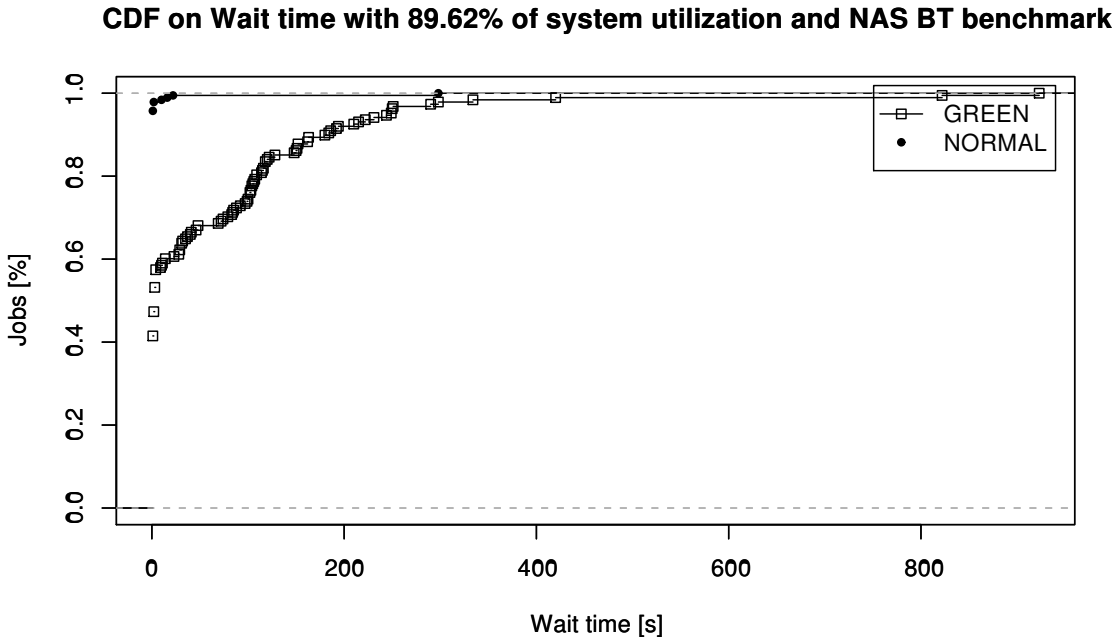


Figure 6.8: Cumulated Distribution function on Wait time for 89.62% of system utilization and NAS BT benchmark with OAR

other. Hence the exact same workload trace submitted to the same cluster with different resource management systems can lead to different green management decisions.

Figure 6.10 show us the strech times for all the executed jobs for each case of RJMS (OAR and SLURM). The strech times represent the turnaround time normalized by the job's actual running time. We can observe that there are nearly no differences between the both figures which explains that the impact upon waiting times where the same between the two platforms.

The new versions of OAR which provide improvements have not been evaluated in this thesis. Nevertheless, we expect that for particular conditions with small-sized jobs there will be an important optimization concerning the waiting times, with perhaps less energy gains for the part of OAR RJMS.

The reason that we use the same workload of 89.62% of utilization is the fact that large utilization rates which stress the system with a lot of submissions can provide us with valuable observations concerning that cannot be seen few jobs or smaller rates. In addition it is rather straightforward that when using energy saving techniques with small utilization rates the energy gain will be much more important than the side-effects upon response times. This will simplify our evaluation.

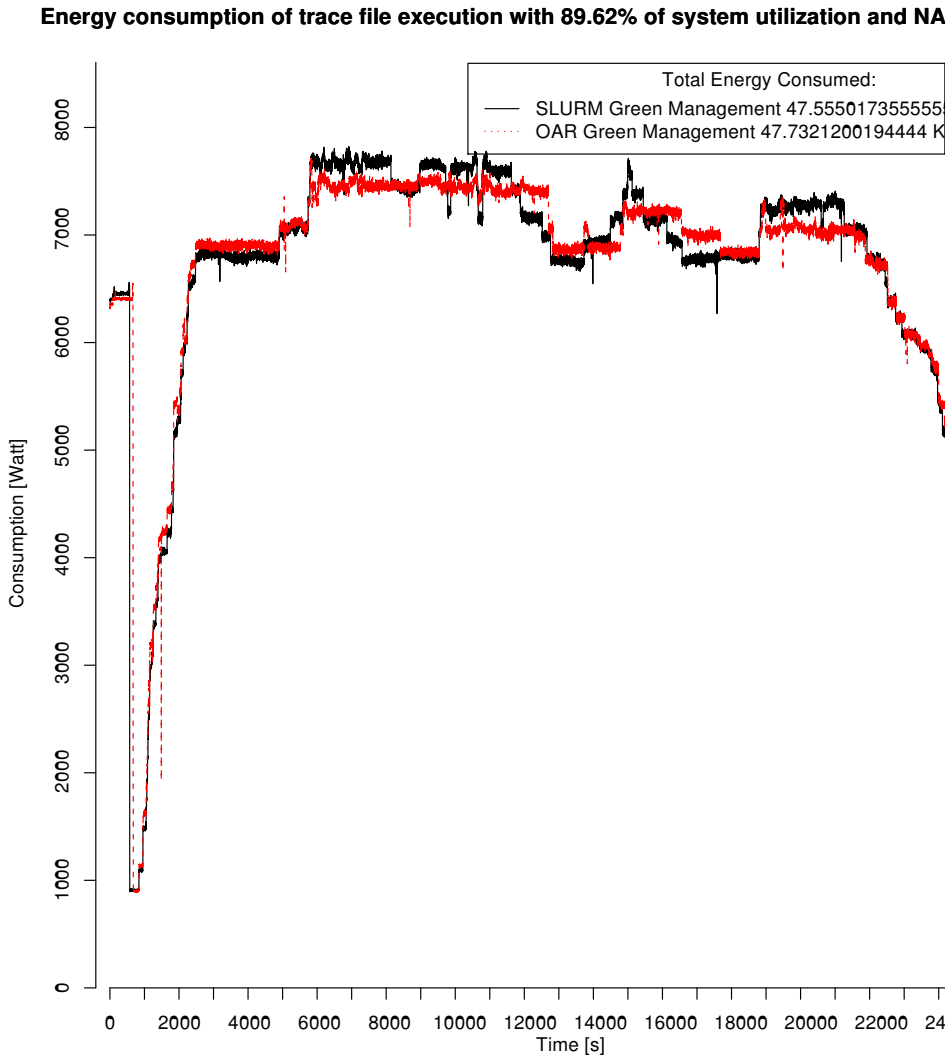


Figure 6.9: Energy Consumption with trace file of 89.62% of system utilization and NAS BT Benchmark upon a 32 nodes(biCPU) cluster with OAR and SLURM Green modes

In contrast our goal is to observe the trade-offs energy consumption versus jobs waiting times in extreme situations even if they are not constantly the case.

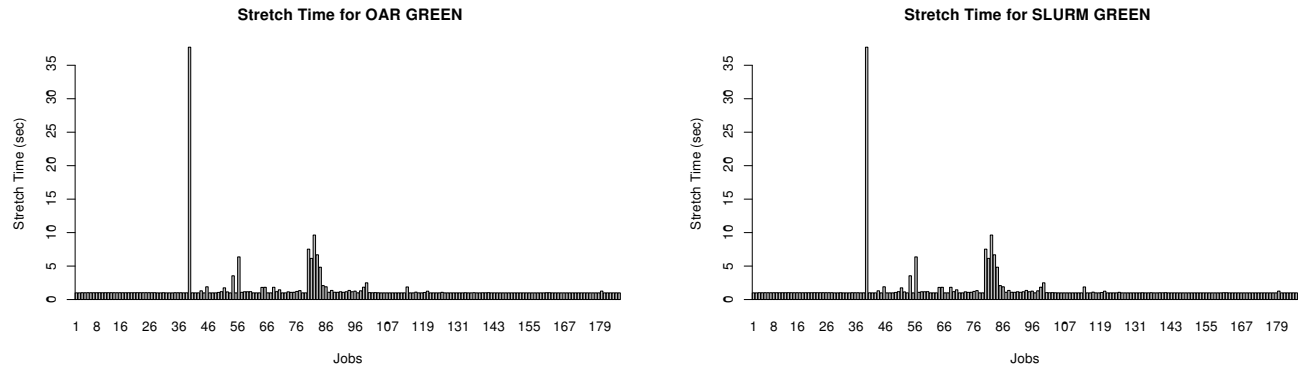


Figure 6.10: Strech times upon a 32 nodes(biCPU) cluster for 89.62%utilization and NAS BT benchmark with OAR and SLURM Green modes

6.4 Supporting DVFS (Frequency Scaling) for user's exploitation

Nowadays users have become more energy conscious and want to be able to control the energy consumption of the cluster during their computation. At the same time, applications can be programmed to provide if a device is not needed or if it can function slowly. Our choices of the hardware devices that can be treated, were defined by the fact that they have to be either parameterized to function slower, consuming less energy, or provide the possibility of a complete power off.

6.4.1 Adapting OAR to provide DVFS techniques

In an effort to allow the users to perform an efficient execution of their applications according to their specific performance needs, we have developped specific options upon the Resource and Job Management System OAR. These options enable the secure, on-the-fly manipulation of the hardware performance.

OAR supports different kind of jobs, like besteffort jobs (lowest priority jobs used for global computing [214]) or deploy type of jobs (used for environment deployment [215]). The implementation of a new powersaving type of job allows the user to control the device power consumption of the computing nodes during their job execution. The open architecture of OAR along with its

flexibility permitted us to integrate this feature with a rather straightforward manner. Unlike most Resource Management Systems, in OAR there is no specific daemon running on the computing nodes of the cluster. Nevertheless, during the execution, the server communicates with every node (participating in the job) where it can obtain root privileges and perform all the demanded power saving modifications. The specific device modifications are stored into the database as different device power states. At the end of the job all computing nodes return to their initial power states.

Our development upon OAR consists of a simple support of CPU frequency scaling and Hard disk spin-down upon modern platforms that provide this kind of hardware treatment. For this we have introduced a new type of jobs called *Powersaving* which allows the user to choose and control the device performance and thus the power consumption of the computing nodes during their job execution.

Currently only CPU and hard-disk speed scaling can be effectuated but other devices support are also planned. The mechanisms make use of specific linux commands like cpufreq-set [6] for cpu frequency scaling and sdparm [7] for hard disk spin down techniques.

6.4.2 Tradeoff DVFS Energy vs Performance

To experiment with this feature we conducted tests using MPI applications to compare the gain in energy consumption when using the different options of powersaving jobs. In particular we make the comparison between four cases: 1)a normal execution (no CPU frequency scaling or HDD spin-down), 2)only CPU frequency scaling, 3)only HDD spin-down, 4)both CPU frequency scaling and HDD spin-down. For this experimentation we use Grid5000 platform and more specifically 9 nodes of Genepi cluster with Intel Xeon E5420 QC 2.5GHz DualCPU-QuadCore, 8GB Memory and Infiniband 20G network. In our experiments we deploy one node as OAR server and 8 computing nodes. We execute NAS NPB benchmarks [66], which are widely used to evaluate the performance of parallel supercomputers. In more detail we execute class D benchmarks with their MPI3.3 implementation[8] and 64 processes (one process per core).

We have calculated the percentage trade-off gains between energy consumption and performance (execution time) for the different powersaving cases normalized with the normal execution for the NPB benchmarks. These results presented in table 6.3 show that the use of powersaving

[6] $http://linux.die.net/man/1/cpufreq-set$
[7] $http://linux.die.net/man/8/sdparm$
[8] $http://www.nas.nasa.gov/Resources/Software/npb\_changes.html$

Method	HDD Spin	CPU Freq	HDD Spin + CPU Freq
Gain %	Energy/Performance	Energy/Performance	Energy/Performance
EP	2.5% / 0%	10.3% / -18.9%	12.2% / -20.5%
SP	1.6% / 0.3%	8.5% / -1.3%	10.2% / -1.5%
BT	2% / -0.4%	9% / -5.4%	10.4% / -5.5%
LU	2.2% / 0.2%	9.5% / -7.6%	11.5% / -10.8%
CG	2% / -0.13%	8.2% / -1.4%	10% / -3.1%
IS	1.4% / 1.5%	6.4% / -1.5%	10% / -7.2%
MG	1.2% / -1.1%	8.2% / -0.5%	9.8% / -3.4%
Overall	1.8% / 0.05%	8.5% / -5.2%	10.5% / -7.4%

Table 6.3: Gain percentage Energy VS Performance (Execution Time)

options achieve good trade-offs between energy reduction and performance loss. Ofcourse the final gain is marginal if someone is interested to have energy reduction with no performance loss. Nevertheless, most of the times some energy benefits are followed by a rather small increase in execution time. Similar results were also observed in previous works [21] which experimented only with CPU frequency scaling. It is suprising to observe how SP,LU and especially IS benchmark present a gain not only in energy reduction but also in performance when HDD spin-down techniques are performed. This strange behaviour can be explained by the presence of system noise and more particular it could be related with the processor cache and TLB (Translation Lookaside Buffer) behaviour [216]. Nevertheless, recent studies that predict disk idle times and use multi-speed disks have shown important energy savings with small loss in performance [217]. Finally the only case that the trade-off energy reduction versus performance loss is not good is in the case of EP benchmark.

The graphics on figure 6.11 show the impact of relation between energy consumption and execution for different options of powersaving jobs for the case of NAS SP benchmark. Each graphic represent 4 different runs of the same benchmark with normal execution (no DVFS technique) and 3 green executions (HDD-spindown, CPU-lowfreq and the merge of both). We can observe that CPU-lowfreq techniques have obviously larger energy consumption gains than the HDD-spindown technique but with impact on waiting times.

The graphics on figure 6.11 show the impact of relation between energy consumption and execution for different options of powersaving jobs for the case of NAS SP benchmark. Each graphic represent 4 different runs of the same benchmark with normal execution (no DVFS technique) and 3 green executions (HDD-spindown, CPU-lowfreq and the merge of both). We can observe that

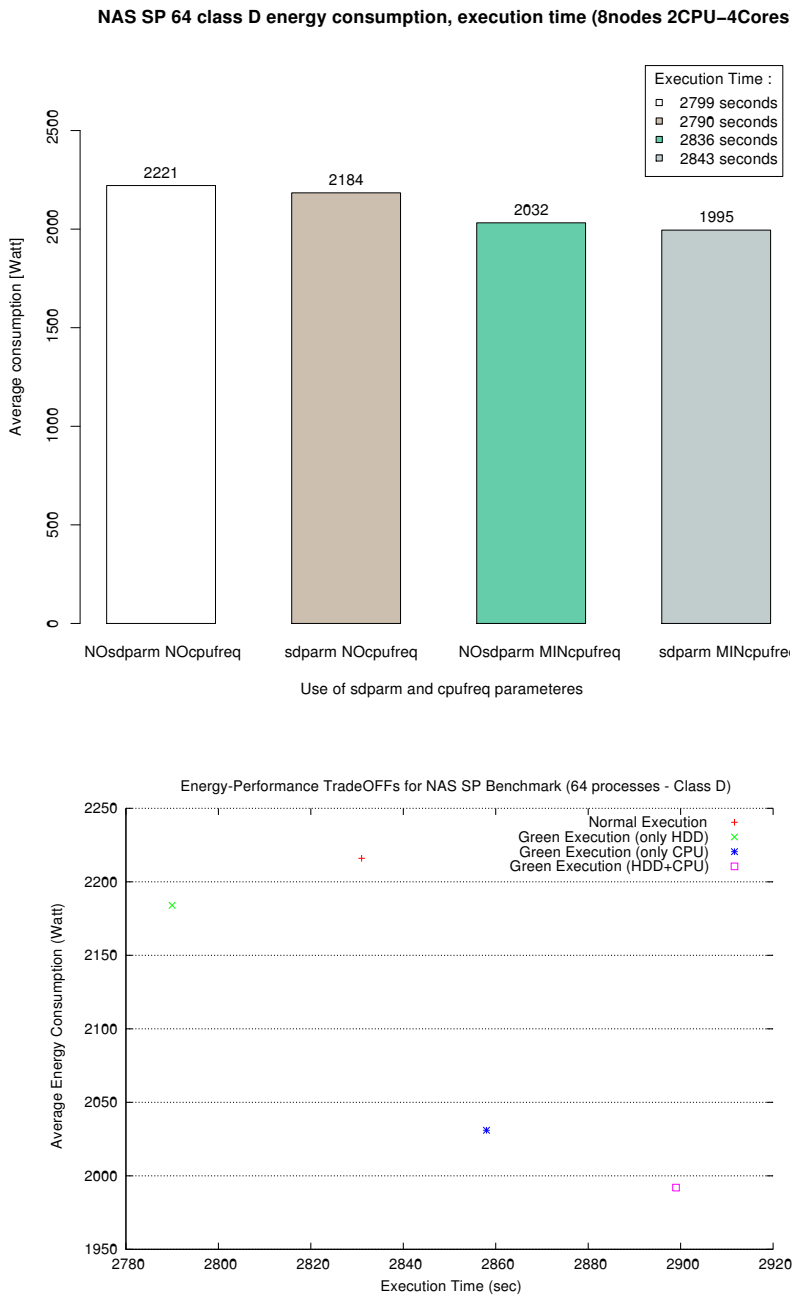


Figure 6.11: NAS SP benchmark executions through PowerSaving jobs with sdparm and cpufreq variations Different representation of the same results for energy-performance tradeoffs

CPU-lowfreq techniques have obviously larger energy consumption gains than the HDD-spindown technique but with impact on waiting times. Similarly with BT and CG, SP benchmark provides

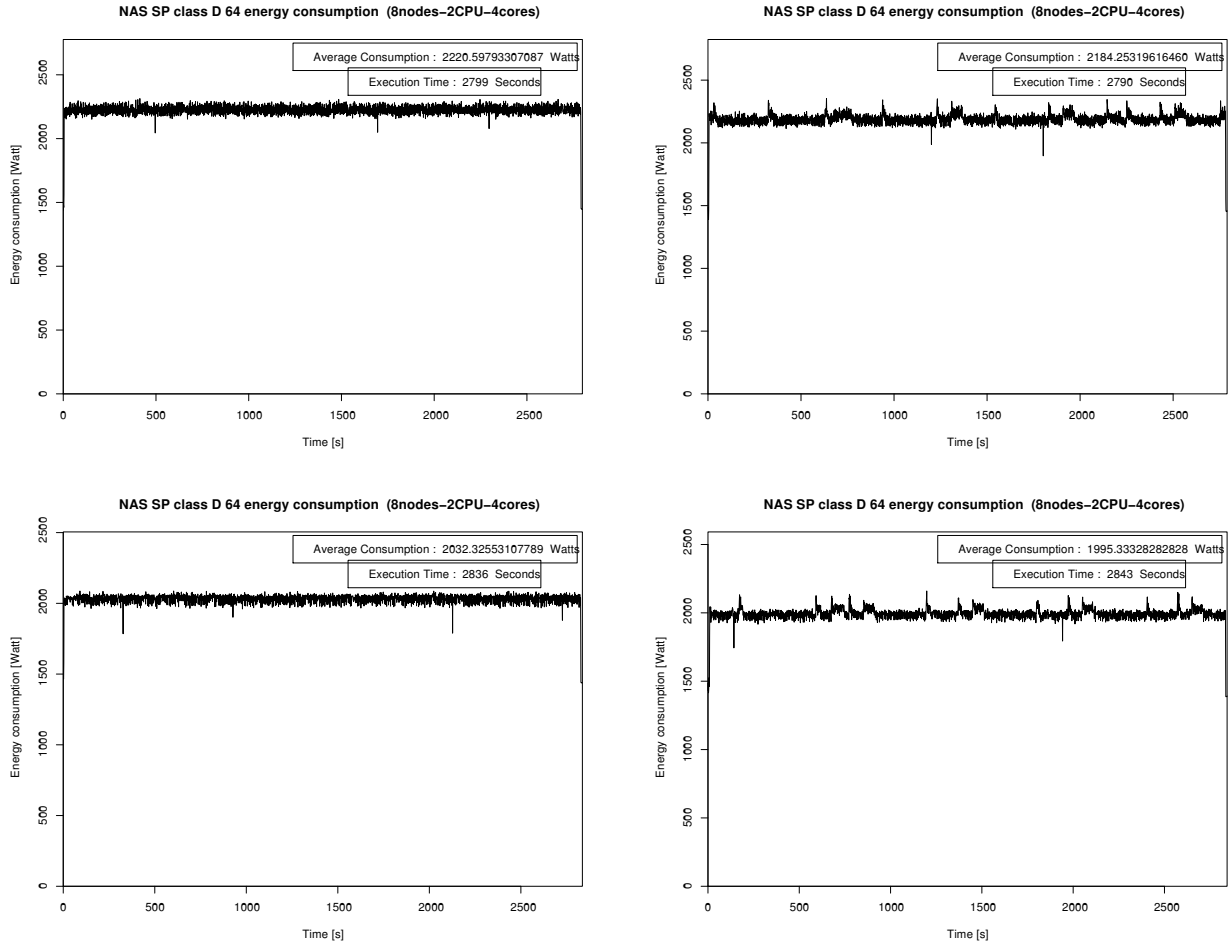


Figure 6.12: Instant energy consumption for NAS SP: Normal(top-right),HDD-spindown(top-left),CPU-lowfreq(bottom-right),HDD-spindown+CPU-lowfreq(bottom-left)

an improvement in performance when HDD-spindown techniques are performed. Finally figure 6.12 provide the instant energy consumption for the execution of NAS SP benchmark and different cases of PowerSaving options.

6.5 Conclusions

In the general context of improving a systems' exploitation, this chapter presents studies around energy efficiency in High Performance Computing and it provides development and experimentation

of resource management techniques which take advantage of unutilized periods of the computing infrastructures in order to perform energy reductions. Our technique considers the idle periods between job executions of the workloads and if specific conditions are met the RJMS triggers actions like Power-OFF upon the idle machines. The technique implies the Power-ON of a machine in case it is needed by a submitted job. The waiting time in this case provides an interesting trade-off to the energy reductions. The beneficial value of the technique partially depends on the various workloads. Experimentation with real workloads presented results upon the trade-off energy reduction and waiting time, along with green management mechanisms comparison between the two resource managers SLURM and OAR. The results showed similar behaviour as expected. However we argue that the new enhanced version of OAR Green Management Mode that dynamically *keeps alive* some machines for interactive or small job rapid utilization may provide smaller waiting times for similar energy reductions. Further experimentation in this context will compare this strategy with the currently proposed ones.

In the same context and in an effort to enable energy reductions on the application level by motivating energy consciousness on users we provided enhanced options through the expression of the job attributes by definining a new type of job (PowerSaving) that enable the Dynamic Voltage and Frequency Scaling of hardware like the CPU and the hard disk. These options are sent as job parameters through the submission of the job and they trigger the particular DVFS commands upon the allocated computing nodes. Particular experiments using the NAS benchmarks showed interesting trade-offs of the energy reduction and the application performance. The support of energy management of other type of hardware(like GPU, network interfaces, memory, etc), through the technique of PowerSaving job parameters, will be provided in future versions of this feature. Furthermore the motivation for energy consioucness upon the users could be also encouraged by providing particular accounting options that consider their jobs energy consumption. By this way the RJMS could provide fairness between jobs and users not only based upon their overall computation time but also upon their overall energy consumption. In this respect we propose to enhance the swf format [24] described on chapter 2 with an additional value that will reflect the energy consumption of the particular job.

Chapter 7

Conclusions and Future Research Directions

7.1 Conclusions

High Performance Computing has made important advances during the last years, driven by the evolutions in microprocessor technologies, high-speed networks and the applications increasing needs for computing power. The Resource and Job Management systems maintain a very important place in this context since they have knowledge of both the system and the applications and they are responsible for actions like scheduling among jobs, matching of resources to user jobs and application execution upon the allocated resources.

The research work presented in this dissertation has resulted into the following publications [25, 26, 27, 28, 29, 30, 31]. In this thesis we have provided a thorough analysis of the internals of the Resource and Job Management systems and we have studied their evolution towards efficiency and scalability. We have effectuated a conceptual comparison among various open-source and commercial RJMS, presented a quantifiable evaluation for their functionalities and a real-scale experimentation to compare particular components like scheduler and launcher efficiency, topology aware placement and energy efficient management techniques. Experimental results revealed significant insights about the characteristics of particular components of some RJMS. Based on these results some system optimization ideas for improvements of OAR and SLURM systems have been proposed.

Related studies upon workload logs of real production platforms showed that regardless the efficiency of a RJMS the computing infastructures, in reality, may suffer of large unutilization

periods. Our motivation is to take advantage of these otherwise idle resources to enable improvements for the exploitation of the cluster.

Initially placed in a context of a lightweight grid where unutilized cluster resources can be aggregated for large-scale grid jobs our studies showed that even if the system utilization may arrive at nearly 100%, the real valuable computations remain low because of interference failures that interupt the jobs without letting them terminate their work. One of the contributions of this thesis was to improve the ratio of valuable versus wasted computations by proposing a fault-tolerant technique based on the checkpoint/restart mechanisms particularly adapted for bag-of tasks applications.

In a similar context upon a single cluster, our motivation is to enable users to make use of malleable jobs that can adapt themselves on system availabilities. Hence an upgrade of the previous approach was proposed by the development of malleability techniques upon a specific RJMS which extends the concept of *besteffort* jobs with *malleable* jobs that consist of a rigid part: to guarantee executions and a besteffort part: to dynamically adapt to resources availabilities. Our results showed significant improvements on system utilization and efficient and valuable system exploitation for both prototype malleability techniques

Nevertheless, high performance computing infrastructures are characterized by important energy consumptions. Hence, we argue that in place of using unutilized resources for additional computations, we could take advantage of those resources to perform energy economies, since they are not used by the principal workload. This alternative system exploitation can lead to important energy reductions and decrease of the computing systems electricity bills.

Hence, combinations of the above techniques, through the use of idle cluster resources, could indeed contribute to attain overall improvements for the exploitation of computing infrastructures, either this is efficient computations or energy reductions. However, there is not one universal solution and there are a lot of parameters that need to be taken into account. In particular the study of workloads of the various systems can reveale valuable information about the particular needs of the specific platform and can enable the correct decisions for improvements of the exploitation of the system. In this thesis we have provided ways to study the behaviour of a system through real-scale reproducible experimental methodologies based upon synthetic or real workload traces. This is another significant contribution of this work, since up to our knowledge there is not a lot of research done in this area of HPC using real-scale experimentation.

The particular *ESP* benchmark was adopted to our experimental methodology and it has been extensively used in our evaluation procedures for resource and job management systems. This

benchmark provides a standard for evaluation of the scheduling and launching capabilities of an RJMS. Under this context we proposed a modified version of *ESP* benchmark (called *TOPO-ESP-NAS*) to evaluate particular resource management features like the efficiency of the topology aware placement techniques.

7.2 Future Research Directions

Concerning future directions of our 3 main proposals, the marginal improvements of the first proposed mechanism upon the lightweight grid context opened directions for future optimizations of this method. In particular, the use of software that can explicitly calculate the duration of the checkpointing operation of an application could enable the more accurate definition of the *grace-time* delay that has to be used on the resource manager. In addition extensions to include migration of checkpoints to different clusters of the same architectures can be taken into account.

The prototype support of malleable jobs upon a resource and job management system will be enhanced to enable the submission of multiple malleable jobs and share the available unutilized resources amongst them based on fair policies like *equipartition*. Moreover, the evaluation of this method needs enhancements by considering various applications and workload traces.

Finally the energy efficient resource management techniques, besides the optimization of energy economies they can be further enhanced to include prediction algorithms to better deal with the unutilized periods and decrease the waiting times of jobs that need the resources These algorithms will take into account information concerning the workloads like peak moments of utilization but also thermodynamical information of the infrastructures like external and internal temperatures of particular areas of the room or hardware components.

Future research directions also include the extension of this methodology with simulation: to abstract particular components of the system and virtualization procedures: to evaluate the system under realistic conditions with smaller number of physical machines and less energy consumption. The results of these initial procedures will be taken into account in the real-scale experimentation phase for better configuration of the whole testbed. Furthermore more enhancements can be made on the ESP benchmark to take into account data staging information during the submission of jobs in order to measure the interfacing capabilities of the RJMS with the different shared file systems.

The multi-level methodology comparison (concepts, functionalities and performance) for Resource and Job Management Systems that we presented in this thesis will be used and updated in order to answer to the RJMS adaptations to the evolving technological and application needs. It

can be extended to include more RJMS and larger scale clusters and number of submitted jobs in order to test the limits of each system.

Finally our study upon the scalability and efficiency issues of particular components of the RJMS like scheduling, topology aware placement, energy-efficiency and fault tolerance can be adapted to fit the needs for the constant hardware and application level evolutions. The continuous study and evolution of these RJMS components is indespensable, especially for the preparation of the near-future passage of High Performance Computing platforms from the petaflop to exaflop scale.

Chapter 8

Annexes

8.1 RJMS Functionalities Comparison

In this section we present a functionalities comparison among some of the most known Resource and Job Management Systems.

General Characteristics / RJMS Software	SLURM	CONDOR	TORQUE	OAR	SGE	MAUI	MOAB	LSF	PBSPro	LoadLeveler
Version	2.2.0-pre8	7.5.3	2.6.0	2.5.0	6.2u5	3.2.16	5.4	7u6	10.2	4.1
Total Number of Code lines	331,282	552,384	237,951	58,010	658,107	113,225	-	-	-	-
Architectures support										
Intel/Opteron x86/x86_64	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
Sun SPARC	NO	YES	NO	NO	YES	NO	YES	YES	YES	YES
PowerPC	NO	YES	NO	YES	YES	NO	YES	YES	YES	YES
Itanium IA64	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
BlueGene	YES	NO	NO	NO	NO	NO	YES	YES	YES	YES
AIX	NO	YES	YES	NO	YES	NO	YES	YES	YES	YES
Operating Systems support										
Windows	NO	YES	NO	NO	YES	YES	YES	YES	YES	YES
Linux	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
MacOS	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
Programming Languages										
C	83.86%	11.2%	86.2%	4%	66.56%	99.83%	-	-	-	-
C++	-	73.3%	-	4%	-	-	-	-	-	-
Perl	0.87%	11.9%	0.72%	66%	-	-	-	-	-	-
Java	-	1%	-	-	24.42%	-	-	-	-	-
Python	0.11%	0.3%	-	-	-	-	-	-	-	-
Sh/Bash	3.28%	-	4.2%	9%	5.79%	-	-	-	-	-
ruby	-	-	-	7%	-	-	-	-	-	-
caml	-	-	-	10%	-	-	-	-	-	-

Table 8.1: General characteristics comparison among various RJMS

Table 8.1 provides a comparison of some general characteristics of RJMS. We can observe how

Way/Level of Support	Representation	Evaluation Points
Advanced Support (Optimized)	YYY	3
Simple Support (Normal)	YY	2
Limited Support (Prototype)	Y	1
No Support	NO	0

Table 8.2: Symbols used in the comparsion table

Resource Management / RJMS Software	SLURM	CONDOR	TORQUE	OAR	SGE	MAUI	MOAB	LSF	PBSPro	LoadLeveler
Resources Treatment										
Hierarchical	YY	YY	YY	YYY	YY	NO	NO	YY	YY	YY
Partitions/Classes/Queues	YY	YY	YY	YY	YY	YY	YY	YYY	YY	YY
Resources Configuration	YY	YY	YY	YY	YY	NO	YY	YYY	YYY	YY
Multi-Cluster Support	YY	YY	NO	YY	YY	NO	YY	YY	YY	YY
Job Launching, Propagation										
Optimized scalable Techniques *Execution control*	YYY	Y	YY	YYY	YY	YY	YY	YY	YY	YY
Simple/Parallel Jobs	YYY	YY	YY	YY	YY	YY	YY	YY	YY	YY
Resources Monitoring/Reporting	YY	YY	YY	YY	YY	YY	YYY	YYY	YYY	YYY
Task Placement										
Resources affinities(CPU,RAM,GPU,..)	YYY	YY	YY	YYY	YY	NO	YY	YY	YYY	YYY
Network Topology Aware Placement	YYY	NO	YY	YY	YY	NO	NO	YY	YY	YY
Hardware internal node Topology Aware Placement	YYY	YY	YY	YY	YY	NO	NO	YY	YY	YY
High Availability	YY	YYY	YY	YY	YYY	NO	YY	YY	YY	YY
Energy Consumption										
Nodes Power ON/OFF	YY	YY	YY	YYY	YY	NO	YY	YY	YY	NO
DVFS Techniques	Y	NO	NO	Y	NO	NO	NO	NO	NO	NO
Thermal load balancing	NO	NO	NO	NO	NO	NO	YYY	YY	NO	NO
Overall Points (/42)	30	22	22	29	25	8	22	29	27	24

Table 8.3: Resource Management Subsystem features comparison among various RJMS

voluminous in programming code are systems like Condor and SGE. In contrary OAR system is implemented by few lines of scripting languages code which prooves its easy development cycle and flexibility. The systems Moab, LSF, PBSPro and LoadLeveler are proprietary so no access to their source code is possible. Then the various functionalities have been seperated into groups depending the particular abstraction layer that they belong according to the conceptual analysis in chapter 3. Hence, each table provides the functionalities of an abstraction layer for each RJMS. Table 8.3 provides a comparison of Resource Management features, table 8.4 of Job Management features and table 8.5 a comparison of Scheduling features. The methodology of comparison is based upon a simple evaluation procedure which is defined by assigning points from 0 until 3, to each functionality according to table 8.2. As we can see in this table, for the sake of finer readability

Job Management / RJMS Software	SLURM	CONDOR	TORQUE	OAR	SGE	MAUI	MOAB	LSF	PBSPro	LoadLeveler
Job Declaration										
Parallel Jobs	YY	YY	YY	YY	YY	YY	YY	YY	YY	YY
Batch Jobs	YY	YY	YY	YY	YY	YY	YY	YY	YY	YY
Interactive Jobs	YY	YY	YY	YY	YY	YY	YY	YY	YY	YY
Array Jobs	YY	YY	YY	YY	YY	YY	YY	YY	YY	YY
Deployment/VM Jobs	NO	YY	YY	YY	YY	YY	YY	YY	YY	YY
Cosystem Jobs	YY	YY	YY	YY	YY	YY	YY	YY	YY	YY
Besteffort Jobs	NO	YY	NO	YY	NO	NO	NO	NO	NO	NO
Moldable Jobs	NO	NO	NO	YY	NO	NO	YY	YY	NO	NO
Evolving Jobs	YY	NO	NO	NO	NO	NO	NO	YY	NO	NO
Malleable Jobs	NO	NO	NO	Y	NO	NO	NO	YY	NO	NO
Job Control										
Reprioritization	YY	YY	YY	YY	YY	YY	YY	YY	YY	YY
Signaling	YY	YYY	YY	YY	YY	YY	YY	YY	YYY	YY
Prolog/Epilog scripts	YY	YY	YY	YY	YY	YY	YY	YY	YY	YY
Start Time estimation	YY	NO	NO	YY	YY	NO	YY	YY	YY	YY
Monitoring										
Vizualization	YY	YY	YY	YYY	YYY	NO	YYY	YYY	YYY	YY
Authentication										
Secure communication	YY	YYY	YY	YY	YY	YY	YY	YY	YY	YY
Kerberos support	YY	YY	YY	NO	YY	NO	NO	YY	YY	YY
Quality Of Services										
Accounting/Reporting	YY	YY	YY	YY	YYY	YY	YYY	YYY	YYY	YY
Application Checkpoint/Restart	YY	YYY	YY	YY	YY	YY	YY	YY	YY	YY
System Checkpoint/Restart	YY	YYY	YY	Y	YY	NO	YY	YY	YY	YY
Suspend/Resume	YY	YY	YY	YY	YY	YY	YY	YY	YY	YY
Data Staging	NO	YY	YY	Y	Y	NO	YY	YY	YY	YY
Interfacing										
MPI Libraries	YYY	YY	YY	YY	YY	NO	YY	YY	YY	YY
Graphical GUI	YY	YYY	YY	YY	YYY	NO	YYY	YYY	YYY	YY
DRMAA API	NO	YY	YY	Y	YYY	NO	YY	YY	YY	NO
Web Services API	NO	YY	NO	YY	YY	NO	YYY	YY	YY	NO
Grids/Clouds	YY	YYY	YY	YY	YYY	NO	YYY	YYY	YY	NO
Overall Points (/78)	41	52	42	47	50	26	51	56	50	40

Table 8.4: Job Management Subsystem features comparison among various RJMS

we use a representation with letters where value `NO` reflects that the feature is not supported and is assigned with 0 evaluation points. The level of support from *Prototype* to *Normal* and finally to *Optimized* is represented, in the comparison tables, by `Y`, `YY` or `YYY` which represent the level of support of the RJMS for the particular feature, reflecting evaluation points from 1 to 3.

Due to lack of space we cannot enter into detail upon each functionality and how the score is attributed for each RJMS. Nevertheless some examples are provided in order to make clearer the evaluation procedure. For example in the case of the support of *Malleable Jobs* functionality of table 8.4, most of the systems do not support it, except LSF which provides a normal support and OAR which proposes a prototype implementation for the support of this feature. In table 8.3, the support of *Nodes Power ON/OFF* feature is basically supported by most of the RJMS except

Scheduling / RJMS Software	SLURM	CONDOR	TORQUE	OAR	SGE	MAUI	MOAB	LSF	PBSPro	LoadLeveler
Scheduling Algorithms										
FIFO	YY	YY	YY	YY	YY	YY	YY	YY	YY	YY
Backfill	YY	NO	YY	YY	YY	YY	YY	YY	YY	YYY
Fairshare	YY	YY	NO	YY	YY	YY	YY	YY	YY	YY
Preemption	YY	YY	YY	NO	YYY	YY	YY	YYY	YY	YY
Multi-Priority	YYY	YY	YY	YY	YYY	YY	YYY	YYY	YYY	YY
Gang-Scheduling	YY	NO	NO	NO	NO	NO	NO	NO	NO	NO
TimeSharing	NO	NO	NO	YY	NO	NO	NO	NO	NO	NO
Queues Management										
External Scheduler support	YYY	YY	YY	YYY	Y	NO	NO	NO	YY	YY
Scheduler per queue	NO	YY	NO	YY	YY	YY	YY	YY	NO	YY
Advanced Reservations	YY	YY	NO	YY	YY	YY	YY	YY	YY	YY
Application Licenses	YY	YY	NO	YY	YY	YY	YY	YYY	YY	YY
Overall Points (/33)	20	16	10	19	19	16	18	19	17	19

Table 8.5: Scheduling Subsystem features comparison among various RJMS

Overall Evaluation / RJMS Software	SLURM	CONDOR	TORQUE	OAR	SGE	MAUI	MOAB	LSF	PBSPro	LoadLeveler
Overall Resource Management (/10)	7.1	5.2	5.2	6.9	5.9	1.9	5.2	6.9	6.4	5.7
Overall Job Management (/10)	5.1	6.5	5.1	5.5	6	3.1	6.1	6.8	6	4.9
Scheduling (/10)	6	5.3	3	5.7	5.7	5.5	5.7	5.7	5.1	5.7
Overall Evaluation Points (/10)	**6.2**	5.7	5.1	**6**	**5.9**	3.4	5.5	**6.4**	5.8	5.4

Table 8.6: Overall Evaluation comparison results among various RJMS

MAUI and Loadleveler which do not have publically available descriptions concerning this feature. However OAR provide an advanced support for this feature. While it powers OFF nodes like the other systems under specific conditions, when they are not utilized, it can dynamically keep some nodes always ON in order to keep the waiting times for small interactive jobs as low as possible. Another example, in table 8.5, the support of *Preemption* Scheduling Policy is implemented upon all systems except OAR while SGE and LSF provide an optimized version of this support. SGE proposes a slotwise preemption which improves the simple preemption with bestfit algorithms. LSF provide methods for avoiding the over-preemption of parallel jobs by reducing the number of jobs that are preempted in order to execute one larger higher priority job.

Our comparison method may provide an important failure rate because of various reasons. First of all, systems may not provide analytical open documentations (commercial RJMS case) and even if those exist they may not describe the internals of the system making the analysis difficult and the evaluation uncertain. The systems evolve rapidly so even if we provide results about a particular version new versions of the software may come out once a month which make the results easily

changeable. Finally, the choice of the particular features for comparison is made according to our point of views and tries to reflect the general current needs for Resource and Job Management Systems for High Performance Computing.

8.2 Dynamic MPI Malleability experiments

The graphs represent our measurements concerning Dynamic MPI Malleability technique of chapter 5. The gain of malleable jobs versus moldable-besteffort jobs is obvious. The white vertical lines of figures represent time between the finalization of a job and the starting of the another. After 5500 sec. the execution of malleable jobs start to be influenced by jobs from the normal workload (rigid jobs). Thus, the white lines (empty spaces) also mean that the OAR does not assign processors that will be soon required by normal jobs.

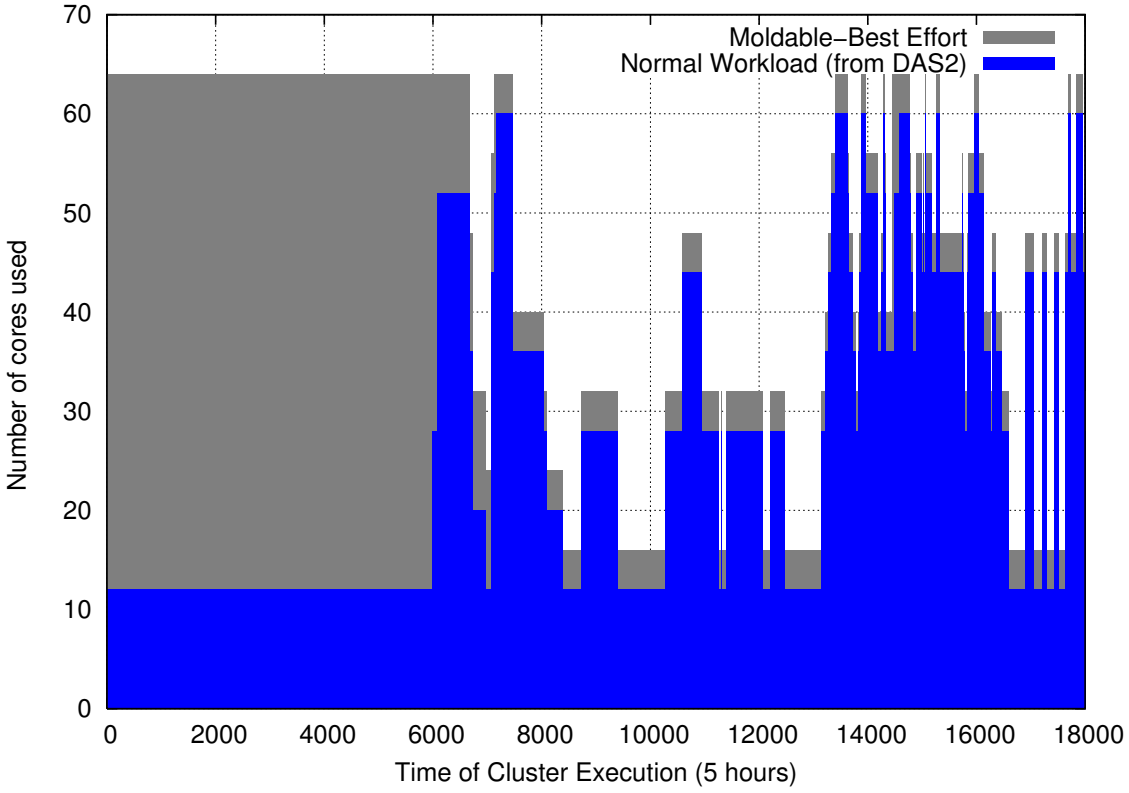


Figure 8.1: Moldable-BestEffort jobs executing upon free cluster resources: 40/% of normal workload utilization and 32/% of Moldable-BestEffort jobs system exploitation of the 60/% that remains free.

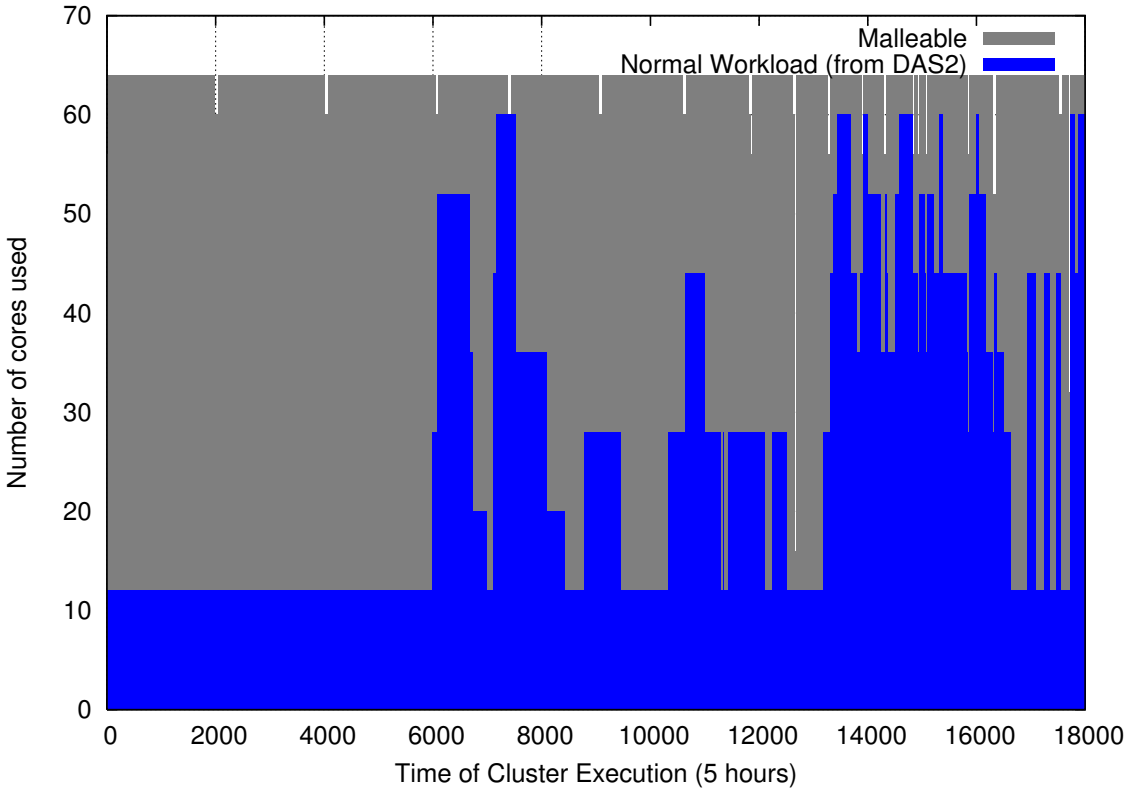


Figure 8.2: Malleable jobs executing upon free cluster resources: 40/% of normal workload utilization and 58/% of Malleable jobs system exploitation of the 60/% that remains free.

Bibliography

[1] Jens Gustedt, Emmanuel Jeannot, and Martin Quinson, "Experimental Validation in Large-Scale Systems: a Survey of Methodologies," 2009.

[2] William TC Kramer, *PERCU: A Holistic Method for Evaluating High Performance Computing Systems*, Ph.D. thesis, EECS Department, University of California, Berkeley, Nov 2008.

[3] "TOP500 Supercomputer Sites," http://www.top500.org/.

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

[5] Hui Li, David L. Groep, and Lex Wolters, "Workload Characteristics of a Multi-cluster Supercomputer," in *JSSPP*, Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, Eds. 2004, vol. 3277 of *Lecture Notes in Computer Science*, pp. 176–193, Springer.

[6] "http://gwa.ewi.tudelft.nl/pmwiki/pmwiki.php?n=Workloads.Overview," .

[7] "Green500 Energy Efficient Supercomputers," http://www.green500.org/.

[8] Cosimo Anglano and Massimo Canonico, "Scheduling algorithms for multiple Bag-of-Task applications on Desktop Grids: A knowledge-free approach," in *IPDPS*. 2008, pp. 1–8, IEEE.

[9] David P. Anderson, "BOINC: A System for Public-Resource Computing and Storage," in *GRID*, Rajkumar Buyya, Ed. 2004, pp. 4–10, IEEE Computer Society.

[10] Franck Cappello, Samir Djilali, Gilles Fedak, Thomas Hérault, Frédéric Magniette, Vincent Néri, and Oleg Lodygensky, "Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid," *Future Generation Comp. Syst*, vol. 21, no. 3, pp. 417–437, 2005.

[11] Walfredo Cirne, Francisco Vilar Brasileiro, Nazareno Andrade, Lauro Costa, Alisson Andrade, Reynaldo Novaes, and Miranda Mowbray, "Labs of the World, Unite!!!," *J. Grid Comput*, vol. 4, no. 3, pp. 225–246, 2006.

[12] Ali Raza Butt, Rongmei Zhang, and Y. Charlie Hu, "A Self-Organizing Flock of Condors," in *SuperComputing*. 2003, p. 42, ACM.

[13] Dror G. Feitelson and Larry Rudolph, "Toward Convergence in Job Schedulers for Parallel Supercomputers," in *Job Scheduling Strategies for Parallel Processing*, pp. 1–26. Springer-Verlag, 1996.

[14] Renaud Lepère, Denis Trystram, and Gerhard J. Woeginger, "Approximation Algorithms for Scheduling Malleable Tasks Under Precedence Constraints.," *Int. J. Found. Comput. Sci.*, vol. 13, no. 4, pp. 613–627, 2002.

[15] Sheikh K. Ghafoor, *Modeling of an adaptive parallel system with malleable applications in a distibuted computing environment*, Ph.D. thesis, 2007, Supervisor-Banicescu, Ioana.

[16] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso, "Power provisioning for a warehouse-sized computer," in *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, New York, NY, USA, 2007, pp. 13–23, ACM.

[17] Jo Ebergen, Jonathan Gainsley, and Paul Cunningham, "Transistor Sizing: How to Control the Speed and Energy Consumption of a Circuit," *Asynchronous Circuits and Systems, International Symposium on*, vol. 0, pp. 51–61, 2004.

[18] W. Feng and T. Scogland, "The Green500 List: Year one," in *IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, Washington, DC, USA, 2009, pp. 1–7, IEEE Computer Society.

[19] Anne-Cécile Orgerie, Laurent Lefèvre, and Jean-Patrick Gelas, "Save Watts in your Grid: Green Strategies for Energy-Aware Framework in Large Scale Distributed Systems," in *14th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, Melbourne, Australia, Dec. 2008.

[20] Anne-Cécile Orgerie, Laurent Lefèvre, and Jean-Patrick Gelas, "Chasing Gaps between Bursts : Towards Energy Efficient Large Scale Experimental Grids," in *PDCAT 2008 : The*

Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies, Dunedin, New Zealand, Dec. 2008.

[21] Vincent W. Freeh, David K. Lowenthal, Feng Pan, Nandini Kappiah, Rob Springer, Barry L. Rountree, and Mark E. Femal, "Analyzing the Energy-Time Trade-Off in High-Performance Computing Applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 835–848, 2007.

[22] Barry Rountree, David K. Lownenthal, Bronis R. de Supinski, Martin Schulz, Vincent W. Freeh, and Tyler Bletsch, "Adagio: making DVS practical for complex HPC applications," in *ICS '09: Proceedings of the 23rd international conference on Supercomputing*, New York, NY, USA, 2009, pp. 460–469, ACM.

[23] Eitan Frachtenberg and Uwe Schwiegelshohn, "New Challenges of Parallel Job Scheduling," in *Job Scheduling Strategies for Parallel Processing*, Eitan Frachtenberg and Uwe Schwiegelshohn, Eds., pp. 1–23. Springer Verlag, 2007, Lect. Notes Comput. Sci. vol. 4942.

[24] Steve J. Chapin, Walfredo Cirne, Dror G. Feitelson, James Patton Jones, Scott T. Leutenegger, Uwe Schwiegelshohn, Warren Smith, and David Talby, "Benchmarks and Standards for the Evaluation of Parallel Job Schedulers," in *Job Scheduling Strategies for Parallel Processing, 13th IPPS/10th SPDP'99 Workshop (5th JSSPP'99)*, Dror G. Feitelson and Larry Rudolph, Eds., San Juan, Puerto Rico, USA, Apr. 1999, vol. 1659 of *Lecture Notes in Computer Science (LNCS)*, pp. 67–90, Springer-Verlag (Berlin).

[25] Yiannis Georgiou, Olivier Richard, and Nicolas 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*, Washington, DC, USA, 2007, pp. 279–286, IEEE Computer Society.

[26] Yiannis Georgiou, Nicolas Capit, Bruno Bzeznik, and Olivier Richard, "Simple, fault tolerant, lightweight grid computing approach for bag-of-tasks applications.," 2008, Short paper and Presentation in 3rd EGEE User Forum, Clermont Ferrand,France.

[27] Marcia C. Cera, Yiannis Georgiou, Olivier Richard, Nicolas Maillard, and Philippe O. A. Navaux, "Supporting MPI Malleable Applications upon the OAR Resource Manager," 2009, Short paper and poster in COLIBRI Informatics Conference, Brazil.

[28] Márcia C. Cera, Yiannis Georgiou, Olivier Richard, Nicolas Maillard, and Philippe Olivier Alexandre Navaux, "Supporting Malleability in Parallel Architectures with Dynamic CPUSETsMapping and Dynamic MPI," in *ICDCN*, 2010, pp. 242–257.

[29] Georges Da Costa, Marcos Dias de Assunção, Jean-Patrick Gelas, Yiannis Georgiou, Laurent Lefèvre, Anne-Cécile Orgerie, Jean-Marc Pierson, Olivier Richard, and Amal Sayah, "Multi-facet approach to reduce energy consumption in clouds and grids: the GREEN-NET framework," in *e-Energy*, 2010, pp. 95–104.

[30] Georges Da Costa, Jean-Patrick Gelas, Yiannis Georgiou, Laurent Lefèvre, Anne-Cécile Orgerie, Jean-Marc Pierson, Olivier Richard, and K. Sharma, "The GREEN-NET framework: Energy efficiency in large scale distributed systems," in *IPDPS*, 2009, pp. 1–8.

[31] Georges Da-Costa, Jean-Patrick Gelas, Yiannis Georgiou, Laurent Lefèvre, Anne-Cécile Orgerie, Jean-Marc Pierson, and Olivier Richard, "The GREEN-NET approach for supporting energy efficient solutions in Grids," Short paper and Poster in Renpar 2009 : French Meeting in Parallelism, Sept. 2009.

[32] Anthony Sulistio, Chee Shin Yeo, and Rajkumar Buyya, "A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools," *Softw, Pract. Exper*, vol. 34, no. 7, pp. 653–673, 2004.

[33] NS2, "The Network Simulator," http://nsnam.isi.edu/nsnam.

[34] Jordi Pujol Ahulló and Pedro García López, "PlanetSim: An Extensible Simulation Tool for Peer-to-Peer Networks and Services," in *Peer-to-Peer Computing*, Henning Schulzrinne, Karl Aberer, and Anwitaman Datta, Eds. 2009, pp. 85–86, IEEE.

[35] Alberto Montresor and Márk Jelasity, "PeerSim: A Scalable P2P Simulator," in *Peer-to-Peer Computing*, Henning Schulzrinne, Karl Aberer, and Anwitaman Datta, Eds. 2009, pp. 99–100, IEEE.

[36] Rajkumar Buyya and M. Manzur Murshed, "GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, pp. 1175–1220, 2002.

[37] Henri Casanova, Arnaud Legrand, and Martin Quinson, "SimGrid: a Generic Framework for Large-Scale Distributed Experiments," in *10th IEEE International Conference on Computer Modeling and Simulation*, Mar. 2008.

[38] Kashi Venkatesh Vishwanath, Amin Vahdat, Ken Yocum, and Diwaker Gupta, "ModelNet: Towards a DataCenter Emulation Environment," in *Peer-to-Peer Computing*, Henning Schulzrinne, Karl Aberer, and Anwitaman Datta, Eds. 2009, pp. 81–82, IEEE.

[39] emulab, "http://www.emulab.net/," .

[40] Huaxia Xia, Holly Dail, Henri Casanova, and Andrew A. Chien, "The MicroGrid: Using Online Simulation to Predict Application Performance in Diverse Grid Network Environments," in *CLADE*. 2004, p. 52, IEEE Computer Society.

[41] Louis-Claude Canon and Emmanuel Jeannot, "Wrekavoc: a tool for emulating heterogeneity," in *IPDPS*. 2006, IEEE.

[42] PlanetLAB, "http://www.planet-lab.org/," .

[43] GENI, "Global Environment for network innovations," http://www.geni.net/.

[44] DAS3, "http://www.cs.vu.nl/das3/," .

[45] Franck Cappello and Henri Bal, "Toward an International "Computer Science Grid"," *Cluster Computing and the Grid, IEEE International Symposium on*, vol. 0, pp. 3–12, 2007.

[46] Brice Videau, Corinne Touati, and Olivier Richard, "Toward an Experiment Engine for Lightweight Grids," Apr. 02 2008.

[47] Walfredo Cirne and Francine Berman, "A Comprehensive Model of the Supercomputer Workload," in *4th Workshop on Workload Characterization*, Dec. 2001, pp. 140–148.

[48] Walfredo Cirne and Francine Berman, "A Model for Moldable Supercomputer Jobs," in *Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS-01)*, Los Alamitos, CA, 2001, pp. 59–59, IEEE Computer Society.

[49] Eitan Frachtenberg and Dror G. Feitelson, "Pitfalls in Parallel Job Scheduling Evaluation," in *JSSPP*, Dror G. Feitelson, Eitan Frachtenberg, Larry Rudolph, and Uwe Schwiegelshohn, Eds. 2005, vol. 3834 of *Lecture Notes in Computer Science*, pp. 257–282, Springer.

[50] "http://gwa.ewi.tudelft.nl/pmwiki/pmwiki.php?n=Home.GWA," .

[51] Allen B. Downey and Dror G. Feitelson, "The Elusive Goal of Workload Characterization," *Performance Evaluation Rev.*, vol. 26, no. 4, pp. 14–29, Mar. 1999.

[52] Adrian T. Wong, Leonid Oliker, William T. C. Kramer, Teresa L. Kaltz, and David H. Bailey, "ESP: A System Utilization Benchmark," in *SC2000: High Performance Networking and Computing. Dallas Convention Center, Dallas, TX, USA, November 4–10, 2000*, ACM, Ed., pub-ACM:adr and pub-IEEE:adr, 2000, pp. 52–52, ACM Press and IEEE Computer Society Press.

[53] Adrian T. Wong, Leonid Oliker, William T. C. Kramer, Teresa L. Kaltz, and David H. Bailey, "System Utilization Benchmark on the Cray T3E and IBM SP," in *Job Scheduling Strategies for Parallel Processing, IPDPS 2000 Workshop, (6th JSSPP 2000)*, Dror G. Feitelson and Larry Rudolph, Eds., Cancun, Mexico, May 2000, vol. 1911 of *Lecture Notes in Computer Science (LNCS)*, pp. 56–67, Springer-Verlag (New York).

[54] Dror G. Feitelson, "A Critique of ESP," in *JSSPP*, Dror G. Feitelson and Larry Rudolph, Eds. 2000, vol. 1911 of *Lecture Notes in Computer Science*, pp. 68–73, Springer.

[55] Sheeba Prakas and Barry Spielberg, "Effective System Performance Suite on IBM e-server pSeries," Tech. Rep.

[56] NERSC, "Effective System Performance (ESP) Benchmark," http://www.nersc.gov/projects/esp.php.

[57] Dror G. Feitelson, "Metric and Workload Effects on Computer Systems Evaluation," *IEEE Computer*, vol. 36, no. 9, pp. 18–25, 2003.

[58] Dror G. Feitelson, "Experimental Computer Science: The Need for a Cultural Change," .

[59] Nicolas Capit, Georges Da Costa, Yiannis Georgiou, Guillaume Huard, Cyrille Martin, Grégory Mounié, Pierre Neyron, and Olivier Richard, "A batch scheduler with high level components," in *5th Int. Symposium on Cluster Computing and the Grid*, Cardiff, UK, 2005, pp. 776–783, IEEE.

[60] OAR, "Resource and Job Management System," http://oar.imag.fr/.

[61] OARGRID, "OAR Wrapper for Grid level allocation," http://gforge.inria.fr/projects/oargrid/.

[62] Yiannis Georgiou, Julien Leduc, Brice Videau, Johann Peyrard, and Olivier 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.

[63] kadeploy, "Environment Deployment tool," http://gforge.inria.fr/projects/kadeploy.

[64] Dror G. Feitelson, "Experimental Computer Science: The Need for a Cultural Change," .

[65] "http://www.cs.huji.ac.il/labs/parallel/workload/index.html," .

[66] David Bailey, Tim Harris, William Saphir, Rob van der Wijngaart, Alex Woo, and Maurice Yarrow, "The NAS Parallel Benchmarks 2.0," Report NAS-95-020, Numerical Aerodynamic Simulation Facility, NASA Ames Research Center, Mail Stop T 27 A-1, Moffett Field, CA 94035-1000, USA, Dec. 1995.

[67] Jaspal Subhlok, Shreenivasa Venkataramaiah, and Amitoj Singh, "Characterizing NAS Benchmark Performance on Shared Heterogeneous Networks," in *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*, Washington, DC, USA, 2002, p. 91, IEEE Computer Society.

[68] J. Dongarra, J. Bunch, C. Moler, and G. W. Stewart, *LINPACK User's Guide*, SIAM, Philadelphia, PA, 1979.

[69] R. F. Barrett, T. H. F. Chan, Eduardo F. D'Azevedo, E. F. Jaeger, K. Wong, and R. Y. Wong, "Complex version of high performance computing LINPACK benchmark (HPL)," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 5, pp. 573–587, 2010.

[70] Christophe Pinte, Francois Menard, Gaspard Duchene, and Pierre Bastien, "Monte Carlo radiative transfer in protoplanetary disks," 2006.

[71] xionee, "Xionee Workload Trace Analysis, Visualization and Replay Toolkit," https://gforge.inria.fr/projects/xionee/.

[72] Niels Fallenbeck, Hans joachim Picht, Matthew Smith, and Bernd Freisleben, "Xen and the art of cluster scheduling," in *Super Computing 06, Virtualization Workshop*. 2006, pp. 237–244, IEEE Press.

[73] Joseph A. Kaplan and Michael L. Nelson, "A Comparison of Queueing, Cluster and Distributed Computing Systems," NASA TM-109025 (Revision 1), NASA Langley Research Center, Hampton, VA 23681-0001, june 1994.

[74] Mark A. Baker, Geoffrey C. Fox, and Hon W. Yau, "Cluster Computing Review," 1995.

[75] James Patton Jones and Cristy Brickell, "Second Evaluation of Job Queuing/Scheduling Software: Phase 1 Report," Tech. Rep., NASA Ames Research Center, june 1997.

[76] Yonghong Yan and Barbara Chapman, "Comparative Study of Distributed Resource Management Systems SGE, LSF, PBS Pro, and LoadLeveler," 2004.

[77] Tarek A. El-Ghazawi, Kris Gaj, Nikitas A. Alexandridis, Frederic Vroman, Nguyen Nguyen, Jacek R. Radzikowski, Preeyapong Samipagdi, and Suboh A. Suboh, "CONCEPTUAL COMPARATIVE STUDY OF JOB MANAGEMENT SYSTEMS," Nsa lucite, 2001.

[78] Tarek A. El-Ghazawi, Kris Gaj, Nikitas A. Alexandridis, Frederic Vroman, Nguyen Nguyen, Jacek R. Radzikowski, Preeyapong Samipagdi, and Suboh A. Suboh, "A performance study of job management systems," *Concurrency - Practice and Experience*, vol. 16, no. 13, pp. 1229–1246, 2004.

[79] Nicolas Capit, Georges Da Costa, Yiannis Georgiou, Guillaume Huard, Cyrille Martin, Grégory Mounié, Pierre Neyron, and Olivier Richard, "A batch scheduler with high level components," in *5th Int. Symposium on Cluster Computing and the Grid*, Cardiff, UK, 2005, pp. 776–783, IEEE.

[80] Susanne M. Balle and Daniel J. Palermo, "Enhancing an open source resource manager with multi-core/multi-threaded support," in *JSSPP'07: Proceedings of the 13th international conference on Job scheduling strategies for parallel processing*, Berlin, Heidelberg, 2008, pp. 37–50, Springer-Verlag.

[81] sched setaffinity manpages, "http://linuxmanpages.com/man2/schedgetaffinity.2.php," 2006.

[82] CPUSET-BULL, "http://www.bullopensource.org/cpuset/," 2007.

[83] Cgroups, "http://www.mjmwired.net/kernel/Documentation/cgroups.txt," 2008.

[84] François Broquedis, Jérôme Clet-Ortega, Stephanie Moreaud, Nathalie Furmento, Brice Goglin, Guillaume Mercier, Samuel Thibault, and Raymond Namyst, "hwloc: A Generic Framework for Managing Hardware Affinities in HPC Applications," in *PDP*, Marco Danelutto, Julien Bourgeois, and Tom Gross, Eds. 2010, pp. 180–186, IEEE Computer Society.

[85] libtopology INRIA, "http://libtopology.ozlabs.org/," 2008.

[86] PLPA-OpenMPI, "http://www.open-mpi.org/projects/plpa/," 2008.

[87] Abhinav Bhatele and Laxmikant V. Kalé, "An evaluative study on the effect of contention on message latencies in large supercomputers," in *IPDPS*, 2009, pp. 1–8.

[88] Abhinav Bhatele, Eric J. Bohm, and Laxmikant V. Kalé, "Topology aware task mapping techniques: an api and case study," in *PPOPP*, 2009, pp. 301–302.

[89] C. E. Leiserson, "Fat-Trees: Universl Networks for Hardware-Efficient Supercomputing," *IEEE Transactions on Computers*, vol. c-34, no. 10, Oct. 1985.

[90] Javier Navaridas, Jose Antonio Pascual, and José Miguel-Alonso, "Effects of Job and Task Placement on Parallel Scientific Applications Performance," in *PDP*, Didier El Baz, François Spies, and Tom Gross, Eds. 2009, pp. 55–61, IEEE Computer Society.

[91] Abhinav Bhatele and Laxmikant V. Kalé, "Application-specific topology-aware mapping for three dimensional topologies," in *IPDPS*. 2008, pp. 1–8, IEEE.

[92] Javier Navaridas, José Miguel-Alonso, Francisco Javier Ridruejo, and Wolfgang Denzel, "Reducing complexity in tree-like computer interconnection networks," *Parallel Computing*, vol. 36, no. 2-3, pp. 71–85, 2010.

[93] Bilardi and Bay, "An Area Lower Bound for a Class of Fat-Trees," in *ESA: Annual European Symposium on Algorithms*, 1994.

[94] Bay and Bilardi, "Deterministic On-Line Routing on Area-Universal Networks," *JACM: Journal of the ACM*, vol. 42, 1995.

[95] Vijay Subramani, Rajkumar Kettimuthu, Srividya Srinivasan, Jeanette Johnston, and P. Sadayappan, "Selective Buddy Allocation for Scheduling Parallel Jobs on Clusters," in *Proc. 2002 IEEE International Conference on Cluster Computing (4th CLUSTER'02)*, Chicago, IL, USA, Sept. 2002, pp. 107–, IEEE Computer Society.

[96] Yariv Aridor, Tamar Domany, Oleg Goldshmidt, Edi Shmueli, José E. Moreira, and Larry Stockmeier, "Multi-toroidal Interconnects: Using Additional Communication Links to Improve Utilization of Parallel Computers," in *JSSPP*, Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, Eds. 2004, vol. 3277 of *Lecture Notes in Computer Science*, pp. 144–159, Springer.

[97] CiteSeerX, "On Enhancing the Reliability of Job Schedulers," 2008.

[98] Christian Engelmann, Stephen L. Scott, Chokchai Leangsuksun, and Xubin He, "Symmetric Active/Active High Availability for High-Performance Computing System Services: Accomplishments and Limitations," in *CCGRID*, 2008, pp. 813–818.

[99] Christian Engelmann, Stephen L. Scott, Chokchai Leangsuksun, and Xubin He, "Transparent Symmetric Active/Active Replication for Service-Level High Availability," in *CCGRID*, 2007, pp. 755–760.

[100] Kai Uhlemann, Christian Engelmann, and Stephen L. Scott, "JOSHUA: Symmetric Active/Active Replication for Highly Available HPC Job and Resource Management," in *CLUSTER*, 2006.

[101] Weikuan Yu, Jiesheng Wu, and Dhabaleswar K. Panda, "Fast and Scalable Startup of MPI Programs in InfiniBand Clusters," in *HiPC*, 2004, pp. 440–449.

[102] Andy B. Yoo, Morris A. Jette, and Mark Grondona, "SLURM: Simple Linux Utility for Resource Management," in *Job Scheduling Strategies for Parallel Processing*, Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, Eds., pp. 44–60. Springer Verlag, 2003, Lect. Notes Comput. Sci. vol. 2862.

[103] gexec remote execution, "http://www.theether.org/gexec/," 2010.

[104] pdsh remote shell, "https://computing.llnl.gov/linux/pdsh.html," 2010.

[105] Cyrille Martin, Olivier Richard, and Guillaume Huard, "Déploiement adaptatif d'applications parallèles. Algorithme de vol de travail appliqué au déploiement d'applications parallèles sur des grappes de grande taille," *Technique et Science Informatiques*, vol. 24, no. 5, pp. 547–565, 2005.

[106] Taktuk, "Adaptive Execution Deployment," http://taktuk.gforge.inria.fr/.

[107] Benoit Claudel, Guillaume Huard, and Olivier Richard, "TakTuk, adaptive deployment of remote executions," in *HPDC '09: Proceedings of the 18th ACM international symposium on High performance distributed computing*, New York, NY, USA, 2009, pp. 91–100, ACM.

[108] Eitan Frachtenberg, Fabrizio Petrini, Juan Fernández, Scott Pakin, and Salvador Coll, "STORM: lightning-fast resource management," in *SC*, 2002, pp. 1–26.

[109] Eitan Frachtenberg, Fabrizio Petrini, Juan Fernández, and Scott Pakin, "STORM: Scalable Resource Management for Large-Scale Parallel Computers," *IEEE Trans. Computers*, vol. 55, no. 12, pp. 1572–1587, 2006.

[110] Dror G. Feitelson and Larry Rudolph, "Parallel Job Scheduling: Issues and Approaches," in *JSSPP*, Dror G. Feitelson and Larry Rudolph, Eds. 1995, vol. 949 of *Lecture Notes in Computer Science*, pp. 1–18, Springer.

[111] Dror G. Feitelson, Larry Rudolph, Uwe Schwiegelshohn, Kenneth C. Sevcik, and Parkson Wong, "Theory and Practice in Parallel Job Scheduling," in *Job Scheduling Strategies for Parallel Processing, (3rd JSSPP'97), IPPS'97 Workshop*, Dror G. Feitelson and Larry Rudolph, Eds., Geneva, Switzerland, Apr. 1997, vol. 1291 of *Lecture Notes in Computer Science (LNCS)*, pp. 1–34, Springer-Verlag (New York).

[112] Eitan Frachtenberg and Uwe Schwiegelshohn, "New Challenges of Parallel Job Scheduling," in *JSSPP*, 2007, pp. 1–23.

[113] Martin W. Margo, Kenneth Yoshimoto, Patricia Kovatch, and Phil Andrews, "Impact of reservations on production job scheduling," in *JSSPP'07: Proceedings of the 13th international conference on Job scheduling strategies for parallel processing*, Berlin, Heidelberg, 2008, pp. 116–131, Springer-Verlag.

[114] Joseph Skovira, Waiman Chan, Honbo Zhou, and David A. Lifka, "The EASY - LoadLeveler API Project," in *IPPS '96: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, London, UK, 1996, pp. 41–47, Springer-Verlag.

[115] David Talby and Dror G. Feitelson, "Supporting Priorities and Improving Utilization of the IBM SP Scheduler Using Slack-Based Backfilling," in *IPPS/SPDP*. 1999, p. 513, IEEE Computer Society.

[116] Ahuva Mu'alem Weil and Dror G. Feitelson, "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling," *IEEE Trans. Parallel Distrib. Syst*, vol. 12, no. 6, pp. 529–543, 2001.

[117] Avi Nissimov and Dror G. Feitelson, "Probabilistic Backfilling," in *JSSPP*, Eitan Frachtenberg and Uwe Schwiegelshohn, Eds. 2007, vol. 4942 of *Lecture Notes in Computer Science*, pp. 102–115, Springer.

[118] J. K. Ousterhout, "Scheduling Techniques for concurrent systems," in *3rd International Conference on Distributed Computing Systems*, Miami, FL, Oct. 1982, pp. 22–30, IEEE.

[119] Dror G. Feitelson and Morris A. Jette, "Improved Utilization and Responsiveness with Gang Scheduling," in *JSSPP*, Dror G. Feitelson and Larry Rudolph, Eds. 1997, vol. 1291 of *Lecture Notes in Computer Science*, pp. 238–261, Springer.

[120] Aurelien Bouteiller, Hinde-Lilia Bouziane, Thomas Hérault, Pierre Lemarinier, and Franck Cappello, "Hybrid Preemptive Scheduling of Message Passing Interface Applications on Grids," *IJHPCA*, vol. 20, no. 1, pp. 77–90, 2006.

[121] G. Berry, "Preemption in Concurrent Systems," in *Proceedings of Foundations of Software Technology and Theoretical Computer Science*, Rudrapatna K. Shyamasundar, Ed., Berlin, Germany, Dec. 1993, vol. 761 of *LNCS*, pp. 72–93, Springer.

[122] Quinn Snell, Mark J. Clement, and David B. Jackson, "Preemption Based Backfill," in *Job Scheduling Strategies for Parallel Processing, 8th International Workshop, (8th JSSPP'02)*, Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, Eds., Edinburgh, Scotland, UK, July 2002, Revised Papers 2003, vol. 2537 of *Lecture Notes in Computer Science (LNCS)*, pp. 24–37, Springer-Verlag (New York).

[123] Edi Shmueli and Dror G. Feitelson, "Uncovering the Effect of System Performance on User Behavior from Traces of Parallel Systems," in *MASCOTS*. 2007, pp. 274–280, IEEE Computer Society.

[124] Achim Streit, "A Self-Tuning Job Scheduler Family with Dynamic Policy Switching," in *JSSPP*, Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, Eds. 2002, vol. 2537 of *Lecture Notes in Computer Science*, pp. 1–23, Springer.

[125] Eitan Frachtenberg, Dror G. Feitelson, Fabrizio Petrini, and Juan Fernandez, "Adaptive Parallel Job Scheduling with Flexible Coscheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. PDS-16, no. 11, pp. 1066–1077, Nov. 2005.

[126] IBM, "http://www-03.ibm.com/systems/software/adleveler/index.html," .

[127] IBM, "http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/topic/com.ibm.cluster.loadl41j.resmgr.do
.

[128] IBM, "http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/topic/com.ibm.cluster.loadl41j.admin.doc
.

[129] Fang Wang, Hubertus Franke, Marios C. Papaefthymiou, Pratap Pattnaik, Larry Rudolph, and Mark S. Squillante, "A Gang Scheduling Design for Multiprogrammed Parallel Computing Environments," in *IPPS '96: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, London, UK, 1996, pp. 111–125, Springer-Verlag.

[130] Songnian Zhou, Xiaohu Zheng, Jingwen Wang, and Pierre Delisle, "Utopia: A Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems," Tech. Rep., 1993.

[131] Hewlett-Packard Development Company, "HP XC System Software," http://docs.hp.com/en/5991-7400/5991-7400.pdf.

[132] Don Mize1 Robert Stober2 Dave Field1, Deron Johnson1, "Scheduling to Overcome the Multi-Core Memory Bandwidth Bottleneck," .

[133] Don Mize1 Robert Stober2 Dave Field1, Deron Johnson1, "Green HPC Dynamic Power Management in HPC," .

[134] Philip M. Papadopoulos, Mason J. Katz, and Greg Bruno, "NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters," *Cluster Computing, IEEE International Conference on*, vol. 0, pp. 258, 2001.

[135] Moab Scheduler-Cluster Resources, "http://www.clusterresources.com/products/mwm/docs/,"
.

[136] Maui Scheduler-Cluster Resources, "http://www.clusterresources.com/products/maui/docs/mauiadmin.
.

[137] David B. Jackson, Quinn Snell, and Mark J. Clement, "Core Algorithms of the Maui Scheduler," in *JSSPP '01: Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, London, UK, 2001, pp. 87–102, Springer-Verlag.

[138] David B. Jackson, "New Issues and New Capabilities in HPC Scheduling with the Maui Scheduler," .

[139] PBS Works, "http://www.pbsworks.com/," .

[140] Robert L. Henderson, "Job Scheduling Under the Portable Batch System," in *IPPS '95: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, London, UK, 1995, pp. 279–294, Springer-Verlag.

[141] Condor Project, "http://www.cs.wisc.edu/condor/," .

[142] Douglas Thain, Todd Tannenbaum, and Miron Livny, "Distributed computing in practice: the Condor experience," *Concurrency - Practice and Experience*, vol. 17, no. 2-4, pp. 323–356, 2005.

[143] X. Evers, R. Boontje, D. H. J. Epema, and R. Van Dantzig, "Condor Flocking: Load Sharing Between Pools of Workstations," Tech. Rep., Feb. 11 1993.

[144] Michael J. Litzkow, Miron Livny, and Matt W. Mutka, "Condor - A Hunter of Idle Workstations," in *Proceedings of the 8th International Conference on Distributed Computer Systems*, Washington, D.C., June 1988, pp. 104–111, IEEE Press.

[145] Todd Tannenbaum, Derek Wright, Karen Miller, and Miron Livny, "Condor: a distributed job scheduler," pp. 307–350, 2002.

[146] Oracle-Sun Sun Grid Engine Manual, "http://wikis.sun.com/display/gridengine62u6/Home," .

[147] Wolfgang Gentzsch, "Sun Grid Engine: Towards Creating a Compute Power Grid," in *Proc. First IEEE International Symposium on Cluster Computing and the Grid (1st CCGRID'01)*, Brisbane, Australia, May 2001, pp. 35–39, IEEE Computer Society (Los Alamitos, CA).

[148] Daniel Templeton, "Begginners guide to Sun Grid Engine 6.2 Installation and Configuration," .

[149] Torque Resource Manager-Cluster Resources, "http://www.clusterresources.com/products/torque/docs/ .

[150] Brett Bode, David M. Halstead, Ricky Kendall, Zhou Lei, and David Jackson, "The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters," in *Proceedings of the 4th Annual Showcase and Conference (LINUX-00)*, Berkeley, CA, 2000, pp. 217–224, The USENIX Association.

[151] SLURM, "Resource and Job Management System," https://computing.llnl.gov/linux/slurm/.

[152] Grid5000, "Experimental grid platform," http://grid5000.fr.

[153] Joris Bremond, "High Availability Documentation on OAR - Admin Guide," Tech. Rep., 2009.

[154] Jason Duell, "The design and implementation of Berkeley Lab's linux checkpoint/restart," apr 2005.

[155] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, Eds., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan-Kaufmann, 1999.

[156] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of Supercomputer Applications and High Performance Computing*, vol. 11, no. 2, pp. 115–128, 1997, ftp://ftp.globus.org/pub/globus/papers/globus.pdf.

[157] Raphaël et al. Bolze, "Grid'5000: a large scale and highly reconfigurable experimental Grid testbed.," *International Journal of High Performance Computing Applications*, vol. 20, no. 4, pp. 481–494, NOV 2006.

[158] James Frey, Todd Tannenbaum, Miron Livny, Ian T. Foster, and Steven Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," *Cluster Computing*, vol. 5, no. 3, pp. 237–246, 2002.

[159] Ian Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems," 2006.

[160] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer, "SETI-home: An Experiment in Public-Resource Computing," *Comm. ACM*, vol. 45, no. 11, pp. 56–61, Nov. 2002.

[161] Dayi Zhou and Virginia Mary Lo, "WaveGrid: a scalable fast-turnaround heterogeneous peer-based desktop grid system," in *IPDPS*. 2006, IEEE.

[162] Marco Aurélio Stelmar Netto, Rodrigo N. Calheiros, Rafael K. S. Silva, César A. F. De Rose, Caio Northfleet, and Walfredo Cirne, "Transparent Resource Allocation to Exploit Idle Cluster Nodes in Computational Grids," in *eScience*. 2005, pp. 238–245, IEEE Computer Society.

[163] Raissa Medeiros, Walfredo Cirne, Francisco Vilar Brasileiro, and Jacques Philippe Sauvé, "Faults in Grids: Why are they so bad and What can be done about it?," in *GRID*, Heinz Stockinger, Ed. 2003, pp. 18–24, IEEE Computer Society.

[164] George Kola, Tevfik Kosar, and Miron Livny, "Phoenix: Making Data-Intensive Grid Applications Fault-Tolerant," in *GRID*, Rajkumar Buyya, Ed. 2004, pp. 251–258, IEEE Computer Society.

[165] Yinglung Liang, Yanyong Zhang, Anand Sivasubramaniam, Ramendra K. Sahoo, José E. Moreira, and Manish Gupta, "Filtering Failure Logs for a BlueGene/L Prototype.," in *DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)*, Washington, DC, USA, 2005, pp. 476–485, IEEE Computer Society.

[166] Michael Litzkow, Todd Tannenbaum, Jim Basney, and Miron Livny, "Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System," Technical Report CS-TR-1997-1346, University of Wisconsin, Madison, Apr. 1997.

[167] Felix Heine, Matthias Hovestadt, Odej Kao, and Axel Keller, "Provision of Fault Tolerance with Grid-enabled and SLA-aware Resource Management Systems," in *Parallel Computing: Current & Future Issues of High-End Computing*, vol. 33 of *John von Neumann Institute for Computing Series*, pp. 113–120. sep 2005.

[168] Patricio Domingues, Artur Andrzejak, and Luis Moura Silva, "Using Checkpointing to Enhance Turnaround Time on Institutional Desktop Grids," in *E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, Washington, DC, USA, 2006, p. 73, IEEE Computer Society.

[169] C. Anglano, J. Brevik, M. Canonico, D. Nurmi, and R. Wolski, "Fault-aware scheduling for Bag-of-Tasks applications on Desktop Grids," in *Grid Computing, 7th IEEE/ACM International Conference on, Vol., Iss., 28-29 Sept. 2006 Pages:56-63*.

[170] Derrick Kondo, Andrew A. Chien, and Henri Casanova, "Resource Management for Rapid Application Turnaround on Enterprise Desktop Grids," in *SC*. 2004, p. 17, IEEE Computer Society.

[171] Chuliang Weng and Xinda Lu, "Heuristic scheduling for bag-of-tasks applications in combination with QoS in the computational grid," *Future Generation Comp. Syst*, vol. 21, no. 2, pp. 271–280, 2005.

[172] Sriram Sankaran, Jeffrey M. Squyres, Brian Barrett, and Andrew Lumsdaine, "The LAM/MPI checkpoint/restart framework: System-initiated checkpointing," in *in Proceedings, LACSI Symposium, Sante Fe*, 2003, pp. 479–493.

[173] Berkeley Lab Checkpoint/Restart BLCR, "https://ftg.lbl.gov/CheckpointRestart/CheckpointRestart.shtn
.

[174] Nicolas Capit, Georges Da Costa, Yiannis Georgiou, Guillaume Huard, Cy rille Martin, Grgory Mouni, Pierre Neyron, and Olivier Richard, "A batch scheduler with high level components," in *Cluster computing and Grid 2005 (CCGrid05)*, 2005.

[175] Blaise Omer Yenke, Jean-François Méhaut, and Maurice Tchuente, "Scheduling Deadline-Constrained Checkpointing on Virtual Clusters," in *APSCC*, 2008, pp. 257–264.

[176] Christopher R. Anderson, "An Implementation of the Fast Multipole Method without Multipoles," *SIAM Journal on Scientific and Statistical Computing*, vol. 13, no. 4, pp. 923–947, July 1992.

[177] Kelvin K. Droegemeier, Keith Brewster, Ming Xue, Daniel Weber, Dennis Gannon, Beth Plale, Daniel A. Reed, Lavanya Ramakrishnan, Jay Alameda, Robert Wilhelmson, Tom Baltzer, Ben Domenico, Donald Murray, Mohan Ramamurthy, Anne Wilson, Richard D. Clark, Sepideh Yalda, Sara J. Graves, Rahul Ramachandran, John A. Rushing, and Everette Joseph, "Service-Oriented Environments for Dynamically Interacting with Mesoscale Weather," *Computing in Science and Engineering (CiSE)*, vol. 7, no. 6, pp. 12–29, Nov. 2005.

[178] Laxmikant V. Kal, Sameer Kumar, and Jayant DeSouza, "A Malleable-Job System for Timeshared Parallel Machines," *Cluster Computing and the Grid, IEEE International Symposium on*, vol. 0, pp. 230, 2002.

[179] Jan Hungershöfer, "On the Combined Scheduling of Malleable and Rigid Jobs," in *SBAC-PAD*, 2004, pp. 206–213.

[180] Gladys Utrera, Julita Corbalán, and Jesús Labarta, "Implementing Malleability on MPI Jobs," in *13th Int. Conference on Parallel Architectures and Compilation Techniques*. 2004, pp. 215–224, IEEE.

[181] Jens-Michael Wierum Jan Hungershföer, Achim Streit, "Efficient Resource Management for Malleable Applications," Tech. Rep.

[182] Kaoutar El Maghraoui, Travis J. Desell, Boleslaw K. Szymanski, and Carlos A. Varela, "Malleable iterative MPI applications," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 3, pp. 393–413, 2009.

[183] Kaoutar El Maghraoui, Travis J. Desell, Boleslaw K. Szymanski, and Carlos A. Varela, "Dynamic Malleability in Iterative MPI Applications," in *7th Int. Symposium on Cluster Computing and the Grid*. 2007, pp. 591–598, IEEE.

[184] Travis Desell, Kaoutar El Maghraoui, and Carlos A. Varela, "Malleable applications for scalable high performance computing," *Cluster Computing*, vol. 10, no. 3, pp. 323–337, 2007.

[185] J. Buisson, O.O. Sonmez, H.H. Mohamed, and D.H.J. Epema, "Scheduling Malleable Applications in Multicluster Systems," in *Int. Conference on Cluster Computing*. 2007, pp. 372–381, IEEE.

[186] Laxmikant V. Kalé, Sameer Kumar, Jayant DeSouza, Mani Potnuru, and Sindhura Bandhakavi, "Faucets: Efficient Resource Allocation on the Computational Grid," in *Proceedings of the 2004 International Conference on Parallel Processing*, August 2004.

[187] LSF, "Platform LSF: Features and Benefits," http://www.platform.com/grids/platform-lsf/features-benefits.

[188] Cluster Resources, "Dynamic and Malleable Jobs," http://www.clusterresources.com/products/mwm/docs/21.3dynamicjobs.shtml.

[189] David Goodell1 William Gropp2 Jayesh Krishna1 Ewing Lusk1 Pavan Balaji1, Darius Buntinas1 and Rajeev Thakur1, "PMI: A Scalable Parallel Process-Management Interface for Extreme-Scale Systems," .

[190] PMI v2 API, "http://wiki.mcs.anl.gov/mpich2/index.php/PMI_v2_API," .

[191] R. H. Castain, T. S. Woodall, D. J. Daniel, J. M. Squyres, B. Barrett, and G .E. Fagg, "The Open Run-Time Environment (OpenRTE): A Transparent Multi-Cluster Environment for High-Performance Computing," in *Proceedings, 12th European PVM/MPI Users' Group Meeting*, Sorrento, Italy, September 2005.

[192] Richard L. Graham, Sung-Eun Choi, David J. Daniel, Nehal N. Desai, Ronald G. Minnich, Craig E. Rasmussen, L. Dean Risinger, and Mitchel W. Sukalski, "A network-failure-tolerant message-passing system for terascale clusters," *Int. J. Parallel Program.*, vol. 31, no. 4, pp. 285–303, 2003.

[193] Darius Buntinas, George Bosilca, Richard L. Graham, Geoffroy Vallée, and Gregory R. Watson, "A Scalable Tools Communications Infrastructure," in *HPCS*. 2008, pp. 33–39, IEEE Computer Society.

[194] Jeffrey M. Squyres and Andrew Lumsdaine, "A Component Architecture for LAM/MPI," in *Proceedings, 10th European PVM/MPI Users' Group Meeting*, Venice, Italy, September / October 2003, number 2840 in Lecture Notes in Computer Science, pp. 379–387, Springer-Verlag.

[195] William Gropp, Ewing Lusk, and Rajeev Thakur, *Using MPI-2 Advanced Features of the Message-Passing Interface*, The MIT Press, Cambridge, Massachusetts, USA, 1999.

[196] Mrcia Cera, Guilherme Pezzi, Elton Mathias, Nicolas Maillard, and Philippe Navaux, "Improving the Dynamic Creation of Processes in MPI-2," in *13th European PVMMPI Users Group Meeting*, Bonn, Germany, 2006, vol. 4192/2006 of *LNCS*, pp. 247–255.

[197] Xizhou Feng, Rong Ge, and Kirk W. Cameron, "Power and Energy Profiling of Scientific Applications on Distributed Systems," in *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers*, Washington, DC, USA, 2005, p. 34, IEEE Computer Society.

[198] S. Huang and W. Feng, "Energy-Efficient Cluster Computing via Accurate Workload Characterization," in *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, Washington, DC, USA, 2009, pp. 68–75, IEEE Computer Society.

[199] Jo Ebergen, Jonathan Gainsley, and Paul Cunningham, "Transistor Sizing: How to Control the Speed and Energy Consumption of a Circuit," *Asynchronous Circuits and Systems, International Symposium on*, vol. 0, pp. 51–61, 2004.

[200] Jung Ho Ahn, Jacob Leverich, Robert Schreiber, and Norman P. Jouppi, "Multicore DIMM: an Energy Efficient Memory Module with Independently Controlled DRAMs," *IEEE Computer Architecture Letters*, vol. 8, pp. 5–8, 2009.

[201] Taeho Kgil, Shaun D'Souza, Ali G. Saidi, Nathan L. Binkert, Ronald G. Dreslinski, Trevor N. Mudge, Steven K. Reinhardt, and Krisztián Flautner, "PicoServer: using 3D stacking technology to enable a compact energy efficient chip multiprocessor," in *ASPLOS*, 2006, pp. 117–128.

[202] Reinaldo Bergamaschi, Guoling Han, Alper Buyuktosunoglu, Hiren Patel, Indira Nair, Gero Dittmann, Geert Janssen, Nagu Dhanwada, Zhigang Hu, Pradip Bose, and John Darringer, "Exploring power management in multi-core systems," in *ASP-DAC '08: Proceedings of the 2008 Asia and South Pacific Design Automation Conference*, Los Alamitos, CA, USA, 2008, pp. 708–713, IEEE Computer Society Press.

[203] Rahul Nagpal, Rahul Nagpal, Y. N. Srikant, and Y. N. Srikant, "Exploring Energy-Performance Trade-offs for Heterogeneous Interconnect Clustered VLIW Processors," Tech. Rep., In Proc. of Intl. Conf. on High Performance Computing, 2005.

[204] Michael S. Warren, Eric H. Weigle, and Wu-Chun Feng, "High-density computing: a 240-processor Beowulf in one cubic meter," in *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, Los Alamitos, CA, USA, 2002, pp. 1–11, IEEE Computer Society Press.

[205] Min Yeol Lim, Vincent W. Freeh, and David K. Lowenthal, "Adaptive, transparent frequency and voltage scaling of communication phases in MPI programs," in *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, New York, NY, USA, 2006, p. 107, ACM.

[206] Yung-Hsiang Lu and Giovanni de Micheli, "Adaptive Hard Disk Power Management on Personal Computers," in *GLS '99: Proceedings of the Ninth Great Lakes Symposium on VLSI*, Washington, DC, USA, 1999, p. 50, IEEE Computer Society.

[207] Ratnesh K. Sharma, Cullen E. Bash, Chandrakant D. Patel, Richard J. Friedrich, and Jeffrey S. Chase, "Balance of Power: Dynamic Thermal Management for Internet Data Centers," *IEEE Internet Computing*, vol. 9, no. 1, pp. 42–49, 2005.

[208] Justin D. Moore, Jeffrey S. Chase, Parthasarathy Ranganathan, and Ratnesh K. Sharma, "Making Scheduling "Cool": Temperature-Aware Workload Placement in Data Centers," in *USENIX Annual Technical Conference, General Track*, 2005, pp. 61–75.

[209] Georges Da-Costa, Jean-Patrick Gelas, Yiannis Georgiou, Laurent Lefèvre, Anne-Cécile Orgerie, Jean-Marc Pierson, Olivier Richard, and Kamal Sharma, "The GREEN-NET Framework: Energy Efficiency in Large Scale Distributed Systems," in *HPPAC 2009 : High Performance Power Aware Computing Workshop in conjunction with IPDPS 2009*, Rome, Italy, May 2009.

[210] Georges Da-Costa, Marcos Dias de Assuncao, Jean-Patrick Gelas, Yiannis Georgiou, Laurent Lefèvre, Anne-Cécile Orgerie, Jean-Marc Pierson, Olivier Richard, and Amal Sayah, "Multi-facet approach to reduce energy consumption in clouds and grids: The GREEN-NET Framework," in *e-Energy 2010 : First International Conference on Energy-Efficient Computing and Networking*, Passau, Germany, Apr. 2010.

[211] "Google Powermeter project," accessed Nov 1, 2009, http://www.google.org/powermeter/.

[212] "Microsoft Hohm project," accessed Nov 1, 2009, http://www.microsoft-hohm.com/.

[213] Hui Li, David L. Groep, and Lex Wolters, "Workload Characteristics of a Multi-cluster Supercomputer," in *JSSPP*, Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, Eds. 2004, vol. 3277 of *Lecture Notes in Computer Science*, pp. 176–193, Springer.

[214] Yiannis Georgiou, Olivier Richard, and Nicolas 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*, Washington, DC, USA, 2007, pp. 279–286, IEEE Computer Society.

[215] F. Cappello et al, "Grid'5000: A Large Scale, Reconfigurable, Controlable and Monitorable Grid Platform," in *6th IEEE/ACM International Workshop on Grid Computing, Grid'2005*, Seattle, Washington, USA, Nov. 2005.

[216] Collin McCurdy, Alan L. Coxa, and Jeffrey Vetter, "Investigating the TLB Behavior of High-end Scientific Applications on Commodity Microprocessors," in *ISPASS '08: Proceedings of the ISPASS 2008 - IEEE International Symposium on Performance Analysis of Systems and software*, Washington, DC, USA, 2008, pp. 95–104, IEEE Computer Society.

[217] Rajat Garg, Seung Woo Son, Mahmut Kandemir, Padma Raghavan, and Ramya Prabhakar, "Markov Model Based Disk Power Management for Data Intensive Workloads," in *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, Washington, DC, USA, 2009, pp. 76–83, IEEE Computer Society.