A. D. MCCXXIV

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Dottorato di Ricerca in Ingegneria Informatica ed Automatica

Comunità Europea
Fondo Sociale Europeo

CONSUMABILITY ANALYSIS OF BATCH PROCESSING SYSTEMS

FLAVIO FRATTINI

Tesi di Dottorato di Ricerca
(XXVI Ciclo)
Marzo 2014

Il Tutore
Prof. Stefano Russo

Il Coordinatore del Dottorato
Prof. Francesco Garofalo

Dipartimento di Ingegneria Elettrica
e delle Tecnologie dell'Informazione

✉ Via Claudio, 21 - 80125 Napoli - ☎ *[+39]* 081 76 83813 - 🗐 *[+39]* 081 76 83816

CONSUMABILITY ANALYSIS OF BATCH PROCESSING SYSTEMS

By
Flavio Frattini

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
AT
FEDERICO II UNIVERSITY OF NAPLES
VIA CLAUDIO 21, 80125 – NAPOLI, ITALY
MARCH 2014

Per aspera ad astra.

Abstract

The use of large scale processing systems has exploded during the last decade and now they are indicated for significantly contributing to the world energy consumption and, in turn, environmental pollution. Processing systems are no more evaluated only for their performance, but also for how much they consume to perform at a certain level. Those evaluations aim at quantifying the energy efficiency conceived as the relation between a performance metric and a power consumption metric, disregarding the malfunction that commonly happens. The study of a real 500-nodes batch system shows that 9% of its power consumption is ascribable to failures compromising the execution of the jobs. Also fault tolerance techniques, commonly adopted for reducing the frequency of failure occurrences, have a cost in terms of energy consumption.

This dissertation introduces the concept of *consumability* for processing systems, encompassing performance, consumption and dependability aspects. The idea is to have a unified measure of these three main aspects. The consumability analysis is also described. It is performed by means of a hierarchical stochastic model that considers the three aspects simultaneously in the process of evaluating the system efficiency and effectiveness. The analysis represents a solution to system owners and administrators that need to evaluate cost-benefit trade-off during the design, development, testing and operational phases.

The analysis is illustrated for two case studies based on a real batch processing system. The studies provides a set of guidelines for the consumability analysis of other systems and empirically confirm the importance of contemplating dependability jointly with performance and consumption for making processing systems really energy efficient.

Acknowledgements

I sincerely thank my advisor professor Stefano Russo for mentoring me during the PhD and for being a role model since I was an undergraduate. Many thanks to professor Domenico Cotroneo for his insightful suggestions and support, both for research and life. He was tough and mild at the right times, a great professor and a real friend. I am very grateful to professor Marcello Cinque for guiding me from behind the curtains. He played a key role during my doctorate.

I also thank professor Kishor S. Trivedi, mentor during the experience at Duke University. Thanks to all (past and current) colleagues of the Mobilab research group and to Rahul Ghosh, office mate at Duke.

I thank all my family. First, I am sincerely thankful to my parents, who sustained me from the beginning. Ciro, Fabrizio and Lucrezia were essential during each step with their candor. Thanks to Barbara and Ettore for their incitement and for trusting me. Thanks to Marcella, always willing. I deeply thank Peppe, who inclined me towards the studies. Unfortunately, I am still ignorant.

Last but certainly not least, I warmly thank Ilaria for her support and encouragement, and for tolerating most of my faults.

Naples, March 2014.

Table of Contents

List of Tables

List of Figures

Introduction

Large processing systems are always more adopted both by industries and research centers. Examples are Amazon's and Google's data centers, Fermi supercomputer [27] in Italy or Blue Waters [95] and OSC [98] in the U.S. Given the huge number of servers, interconnecting devices, cooling components, those systems are significantly contributing to the world energy consumption which, in turn, causes environmental issues because of the release of carbon dioxide, pollutants, and sulfur into the atmosphere by the coal or oils used to generate electricity. Everyone working in the IT is ethically required to spend some effort in the reduction of the environmental impact of data centers.

Green Computing has then been introduced as the practice of creating or making IT installations environmentally sound, i.e. to reduce their electricity consumption. Nevertheless, *there's many a slip 'twixt the cup and the lip*! *Greening large data centers is not an easy task. There is a huge number of interacting components, several types of load each differently affecting the system, service level agreements to maintain, specific design constraints, and failures that can happen.*

Due to the increasing societal sensibility on green computing, new ways of evaluating large processing systems characteristics are being adopted. Since 2007, the well-known Top500 [127] performance ranking of supercomputers is complemented by the Green500 [58], where the ranking is performed according to the energy efficiency. The focus has been shifted to other performance metrics of interest in order to reduce the ever increasing total

cost of ownership of current and future processing systems. A number of management strategies, such as scheduling and load balancing, have been proposed aiming at reducing energy consumption without affecting the performance. *Energy efficiency* is conceived as the relation between a performance metric and a power consumption metric. By way of example, when using the standard benchmark SPECpower [122], the energy efficiency metric is expressed as *Java operations per Watt*. That is, the pure computation capabilities evaluated in a controlled environment are related to the energy consumption. Results achieved in this way are just the evaluation of hardware characteristics in terms of computing and consumption properties that can be compared to the ones provided in the data sheets. This is far from the actual behavior of a processing system during its operational phase. Specifically, our focus is on the malfunction that can take place. *Unfortunately, failures do happen in processing systems and the larger the system, the larger the chance of failure*, given the greater number of interconnected components. Systems with hundreds of thousands of nodes are expected to have one component failure every 30 minutes [61].

Jobs submitted to batch systems may experience mean times between failures in the order of days or hours, compromising the quality of the service as perceived by the users waiting for the results [91]. Such *failures have a measurable cost in terms of wasted energy.* It suffices to think to a scientific data center running batch jobs, i.e. jobs executed without user interaction. A job may abort during its execution and the whole work done from the beginning up to the abort is lost, and the consumed electric power represents a cost of the failure. In the case of a real 500-nodes system at our University, 9% of the energy consumption is directly imputable to failures, as it will be shown in this dissertation. A fault tolerance strategy, such as checkpointing, can mitigate the waste of energy by restoring the failed job from its last checkpoint. However, *a continuously running fault tolerance mechanism has a cost too, in terms of energy.* Then, *is it better checkpointing the jobs, or letting them abort and be re-issued again?* And, *what is the impact of these choices on*

other aspects of the system, such as performance? Also, *do the management techniques for the energy efficiency have an effect on failures?*

All these aspects are ignored in current scientific literature. As a matter of fact, classical energy efficiency metrics, such as the performance-power ratio, do not take into account energy leakages due to failures or the cost of fault tolerance means. Consider an operator that is going to use a benchmarking tool, like SPECpower, to evaluate the energy efficiency of a datacenter. One could imagine the operator monitoring the system while the benchmark runs. Suppose that the system experiences a failure (e.g., some tasks of the benchmarking are not executed by a set of machines). Likely, the operator throws away results that s/he achieved, solves the problem, and restarts the benchmarking. Results reported to the *administrator* are not truthful. *Failures have to be contemplated; otherwise we would evaluate the efficiency of an ideal system.* The presence of failures is silent in not saturated systems, e.g., working between 10% and 50% of their capacity (as in the majority of cases [109, 132, 113]), since their effects on the throughput are not visible, being largely masked by the great availability of spare resources. In these systems, the energy wasted due to failures remains a hidden cost, while it may represent a non-negligible fraction of the overall power consumption.

Thesis contribution:

In order to properly study the efficiency of a processing system, a new attribute is introduced, the ***consumability***, *which represents the ability of a system to produce useful work in a certain time with energy efficiency, that is to properly exploit incoming (electric) power contemplating the (inevitable) occurrence of failures.*

While common energy efficiency metrics evaluate the relation between performance and consumption without malfunction, consumability encompasses also the consumption due to failures and to possible strategies of fault tolerance implemented in a system. Hence, a

processing system is analyzed taking into account performance, consumption and dependability aspects, and their relations. This joint analysis is named *consumability analysis* and it is performed by means of a stochastic model dealing with the three aspects simultaneously in the process of evaluating system efficiency and effectiveness; this is referred to as *consumability model*. A metric for the consumability is proposed to complement traditional ones. The idea is to explicitly evaluate the amount of electric power spent for failures and their handling, and to quantify the efficiency as the ratio between the electric power usefully consumed to complete jobs and the total power consumption of the system. *The proposed metric unearths hidden costs due to failures and allows one to select the solutions that provide the desired trade-off among performance, energy efficiency, and dependability concerns.*

The proposed consumability analysis is conceived for a generic batch processing system to support the estimation process independently from the specific system. The consumability model is achieved through the hierarchical composition of performance, consumption and failure models defined from the study of batch processing systems, and then validated against the data collected from a real system at the Federico II University of Naples seen as a case study. The critical step in defining the model is the identification of energy consumption dynamics related to different performance levels and possible failures. Inputs for the model are achieved from the study of the system in hand; outputs measure performance, instantaneous power absorption and availability.

System owners and administrators can benefit from the consumability analysis during design, development, testing and operational phases. During design and development stages, the analysis helps size and configure the system towards the best trade-off among performance, consumption and dependability. In the subsequent phases, scheduling or power management strategies as well as dynamic repair strategy parameters can be tuned to maintain

the desired consumability level.

In this thesis, the impact on the consumability of different fault tolerance and management techniques is evaluated. Specifically, job checkpointing, job replication and management (both of workload and power) by means of a Cloud platform are assessed. For fault tolerance, the choice is for the most commonly adopted techniques [24]. As for the cloud, it is mainly based on virtualization, which appears as a largely used technology for greening data centers [66].

Experimental results reveal interesting observations for the studied system corroborating the importance of *consumability*:

- Since resources are not saturated, the 9% energy squandering because of failures is hidden if looking at traditional performance and consumption metrics only.
- Fault tolerance techniques, while lightly affecting performance, may by accounted for up to 52% of the total system energy consumption; hence, the proper selection of the fault tolerance technique and its tuning are essential to make the system consumable.
- Cloud and virtualization can bring consumption but also performance down; workload characterization and job classification are then useful for smart scheduling that leaves performance similar to the case of physical servers while also improving availability.

This dissertation addresses the problem of administrators of evaluating efficiency and effectiveness of processing systems by:

- **Defining the consumability**. Current literature only concerns the relation between performance and consumption for evaluating energy efficiency of processing systems.

The consumability is a new attribute of processing systems, which conceives dependability jointly with performance and energy consumption to determine the ability of a system to produce useful work with a reduced power consumption contemplating the failures that inevitably occur.

- **Providing a procedure for the consumability analysis of batch processing systems**. Existing evaluation approaches for processing systems view the three aspects of performance, consumption and dependability separately. Pure measurement based approaches are costly and time consuming. Based on a composite model encompassing the three aspects and applicable to any batch system, the procedure represents a solution for system administrators who need to carry out a cost-benefit analysis aiming at identifying configurations and management strategies balancing throughput, consumption and availability.

- **Introducing a metric**. Energy efficiency is usually computed as the relation between a performance metric and a consumption metric. To quantify the consumability of a processing system, a new metric is introduced, representing the percentage of the incoming power not wasted for failures.

- **Exemplifying the consumability analysis**. By means of two case studies, all the steps for carrying out the analysis are performed, from the preceding characterization of the system *status quo* which populated the model, up to the identification of performance-consumption-dependability trade-offs.

The two case studies show that administrators may take wrong decisions if not evaluating performance, consumption and dependability aspects simultaneously, demonstrating the importance of the consumability and of the related analysis.

The dissertation is organized as follows.

In **Chapter 1** the problem of the energy efficiency in processing systems is introduced. The chapter classifies and reviews current technologies and techniques adopted for the improvement of efficiency and remarks the neglecting of failures.

In **Chapter 2** the basic concepts of performance, energy consumption and dependability, as well as their relations are discussed. The concept of consumability and the problem of its evaluation are also introduced.

Chapter 3 discusses the possible ways to evaluate processing systems. A scalable model based approach is preferred to measurement based approaches, which are costly and time consuming.

Chapter 4 proposes a solution to the evaluation of techniques and technologies oriented to processing systems. Such a solution considers the mutual relations among the peculiar aspects of the systems. Moreover, a metric to quantify the consumability is introduced.

Chapter 5 presents the consumability analysis of a real batch processing system. The various steps of the proposed solution are performed from model populating to analysis of the results and comparison of the proposed metric with the commonly adopted one.

In **Chapter 6** a further experimental study evaluating the impact on consumability of management strategies based on cloud and virtualization is presented. Results of the last two chapters confirm the importance of the consumability aspect and the effectiveness of the proposed solution.

Chapter 1

Energy Efficiency of Processing Systems

Large data centers are responsible for a significant portion of the total energy consumption of entire countries; therefore, their environmental impact and operating cost due to electricity can not be neglected. Energy efficiency came trough as a main aspect of processing systems, in the last decade. In most of the literature, it is conceived as the relation between performance and consumption. Several techniques and technologies have been proposed trying to improve this relation. They are surveyed in this chapter together with commonly used metrics in the field of energy efficiency of processing systems.

1.1 Problem Overview

Energy efficiency is conceived as a good trade-off between performance and consumption of a component. The concept can be applied to many research fields; we focus on processing systems, for which several keywords and buzzwords have been introduced in the study of energy efficiency. Examples are *sustainable computing*, *green computing*, *green IT*, *green Internet*, *energy aware* -, *energy efficient* - (scheduling, load balancing, etc.). With these keywords many papers related to energy efficiency can be found.

In this field, the main goal is to demonstrate how to reduce the energy consumption of computer systems without affecting performance or, conversely, how to improve the

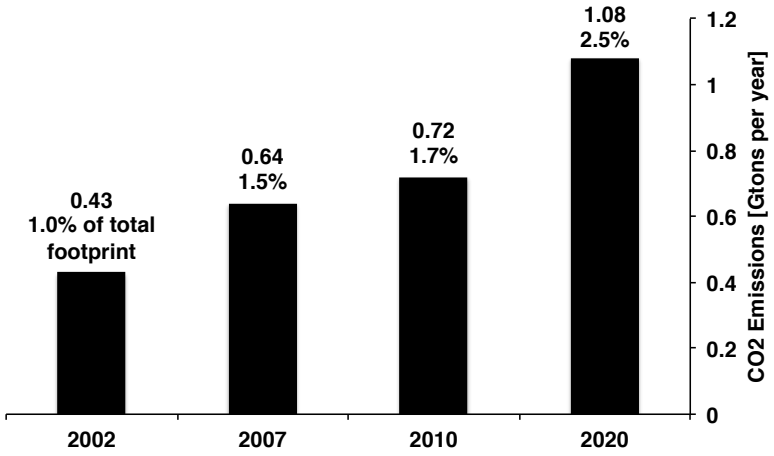


Figure 1.1: Estimate of the carbon footprint of ICT in 2010-2020 decade.

performance without affecting the energy consumption. Large data centers are the ones receiving more attention due to their huge consumption of electricity. The consumption of all the data centers in the U.S. is estimated to exceed 100 billion of kWh [79]. This also arises environmental and ethical questions because of the release of carbon dioxide, pollutants, and sulfur into the atmosphere by the coal or oils used to generate electricity [79, 93]. Figure 1.1 (adapted from [12] and [43]) shows the carbon footprint of ICT infrastructures and devices (data centers, networks, user equipment) in past years and previsions for 2020. The growth expected in the 2010-2020 decade raised many concerns.

Because of these data, governments have been promoting initiatives, projects, and research for reducing the environmental impact of large computer systems. Examples are the "ecodesign requirements for computers, servers and displays" [29] and Intelligent Energy Europe [126] by the European Commission, ICT Footprint supported by the European Commission DG Information Society [44], the Energy Star program by the Environmental

Figure 1.2: Reasons for using eco-responsible practices in IT.

Protection Agency in the U.S. [42], and Top Runner program by the Energy Conservation Center in Japan [41].

Figure 1.2 (adapted from [93]) shows the main reasons for reducing the consumption of IT systems from an interview of 1,500 people. The main objective is to reduce costs, but environmental issues are not disregarded. 55% of interviewed people said being interested in improving the efficiency of the system by better exploiting available resources. Obviously, to the consumption reduction have not to correspond a decrease in system performance.

Wu-Chun Feng was one of the first to introduce the problem of the energy efficiency for large processing systems in a 2003 article entitled "*Making a case for Efficient Supercomputing*" [45]. Feng observed that since 1991 there had been a difference between the improvement in performance of systems running parallel scientific applications (10,000-fold increase) and their performance per watt (which only improved 300-fold) and performance per square foot (65-fold). These results were reprised also in [25] pointing out that *processing systems make a non efficient use of the space and of the electricity*, and banishing the "performance-at-any-cost design mentality", which "ignores supercomputers' excessive

power consumption and need for heat dissipation and will ultimately limit their performance". In the second half of 2000s, the attention on energy efficiency of large computer installations became a diffused problem up to inspiring the foundation of the Green500 ranking, which relates performance and consumption of data centers [120]. The driving motivation was that, if focusing only on performance metrics for evaluating processing systems, we know that they are capable of performing up to trillions of floating-point operations per second, but we neglect that the request of electric power for performing such operations may be prohibitive.

Because of these observations, researchers and practitioners started to figure out *how to improve the performance-consumption trade-off of processing systems.*

An article appeared in Science in March 2013 discusses the methods for improving the energy efficiency for telco operators due to the growing demands of increasing Internet usage [106]. Several techniques and technologies can be applied. Examples are energy efficient management algorithms (e.g., for the network control). A first, intuitive solution is to adopt *switch off-on approaches*; namely, some components are switched off at appropriate times to avoid electricity wasting and switched on again only when necessary. Nevertheless, it is often unfeasible to implement each solution to study its impact. In this perspective, the author highlights the importance of analytical models in order to design green network devices; as an instance, a thermal model of a device can help to estimate its temperature statistics and to decide when starting a strategy for its reduction. Identifying and selecting a technique for reducing the energy consumption of large-scale systems is not an easy

task; creating the support models is not easy as well. In these systems, trend depends on many factors, and understanding and forecasting energy or temperature is somewhat difficult [101]. Hardware, operating system, load balancing, virtualization are all adjustable characteristics of computers; also, once a solution is selected, it is important to tune it in an intelligent and coordinated way to avoid that consumption improvements may affect other characteristics of the system (e.g., performance or readiness in providing the desired service).

1.2 Strategies and Technologies

Many scientific papers discuss ideas, strategies, techniques, or technologies for improving the energy efficiency of computer systems. An overview of possible strategies is presented in [60]. Authors investigate 14 Danish companies and identify three possible strategies. The first one is *storefront*: some companies only focus on complying with legal requirements to create some form of a green image. Other companies focus on the *tuning* of existing resources trying to reduce their consumption and improving the energy efficiency. Sample practices are the implementation of server virtualization and switching off computers. Yet other (few) companies want to fully leverage green IT's potential by *redesigning* processes, identifying new business opportunities enabled by green IT, and changing corporate culture. In this dissertation, the main focus is on the technicalities - such as the tuning or reconfiguration of existing resources - adopted by the management strategies for better exploiting available materials in order to increase the energy efficiency of their cooperation towards the fulfil of some constraints.

In [55], energy efficient approaches are classified depending on static and dynamic aspects of energy consumption. Techniques related to low-power and energy efficient hardware equipment are accounted among the approaches aiming at reducing static consumption; examples are CPUs and power supplies. Dynamic techniques are the ones for which knowledge of current resource utilization and application workloads are used for improving energy efficiency; examples are scheduling and load balancing algorithms. However, we think this classification be misleading. As an instance, there are energy efficient technologies adopted for CPUs that improves also dynamic consumption aspects; on the other side, power management techniques can also reduce the total static consumption of a whole data center, if idle machines are switched off.

We classify the works in the literature on energy efficient processing systems as depicted in Figure 1.3. Table 1.1 reports the referenced works for each class.

Hardware related papers are focused on the realization of *components* that reduce the consumption of each node of a computer system or the consumption of the interconnection of such nodes. In other cases, papers discuss how to improve the *cooling* system in order to reduce related electricity consumption. Although most of the papers in this class are focused on the static consumption of the system, some of them also propose hardware related approaches that take into account the dynamics of the system which depends on executed tasks. Moreover, whatever technique benefits from energy efficient hardware, hence it can not be excluded from the group of dynamic power management. In the case of **software** related papers, programming techniques making a more efficient usage of hardware

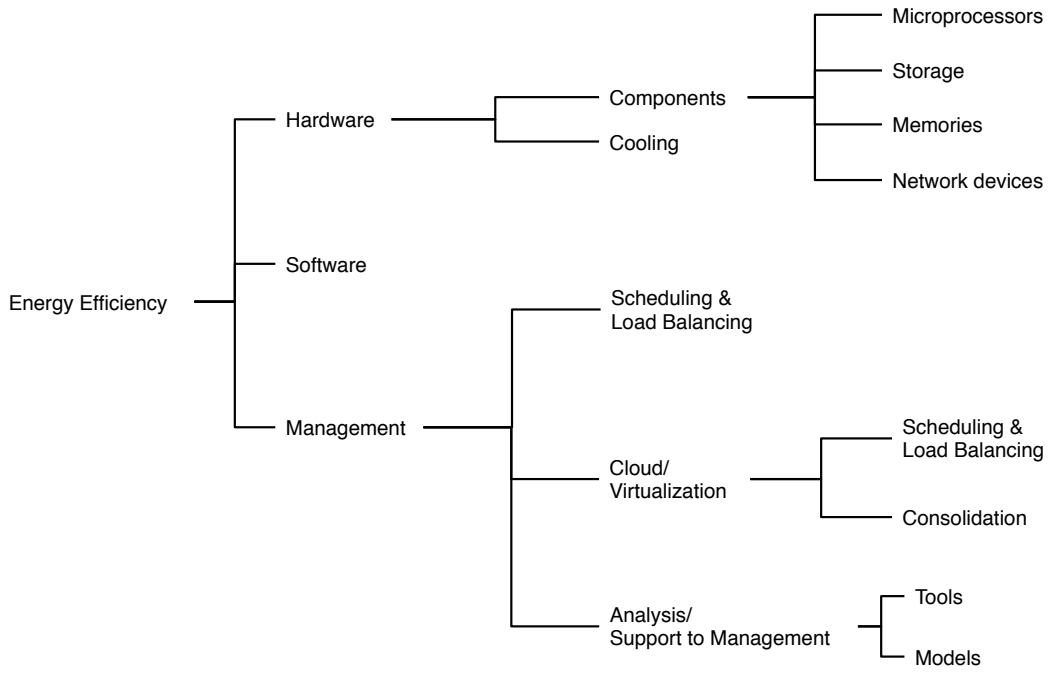


Figure 1.3: Classification of the scientific literature on energy efficiency in data centers.

Table 1.1: References for the identified classes of the scientific literature on energy efficiency.

Category	**References**
Components	[131, 13, 33, 2, 52, 75, 65, 20, 7, 59, 85, 140, 74, 73]
Cooling	[104, 116, 80, 38, 119, 77]
Software	[121, 21, 37, 125, 110, 111, 131, 103, 76]
Scheduling - load balancing	[56, 22, 50, 116, 9, 142, 83, 136]
Cloud computing - virtualization	[57, 81, 86, 133, 137, 92, 82]
Analysis - support to management	[122, 123, 51, 46, 9, 54, 53, 142]
Metrics	[136, 102, 25, 31, 90]

resources are discussed. Most of the effort is made in improving **management** operations. In this case, only dynamic aspects are considered for improving the efficiency of the systems. Examples are the studies about efficient *scheduling* and *load balancing* algorithms. Other papers are about *tools* that support the analysis of large computer systems (e.g., to relate performance and consumption) or about *models* for estimating performance and consumption trends. *Virtualization and cloud computing* are also adopted as a set of techniques and technologies supporting the management of data centers to improve the energy efficiency. Specific scheduling and load balancing algorithms have been developed for virtualized data centers, as well as consolidation strategies for reducing the number of actually used machines.

1.2.1 Hardware

Components

The first effort in reducing the electricity consumption of computer systems has been made on **microprocessors** [131]. Several techniques have been studied both to reduce the consumption during the operational phase (dynamic consumption) and the idle power (leakage power consumption). They can be classified as depicted in Figure 1.4. **Circuit techniques** are related to the low level design of microprocessors (e.g., transistors' reordering, logic gates' restructuring) and to the adoption of low power support components (low power control logic and flip flops). Other **hardware adaptations** are related to the architectural organization of caches, queues and registers. Interconnections heavily affect the power

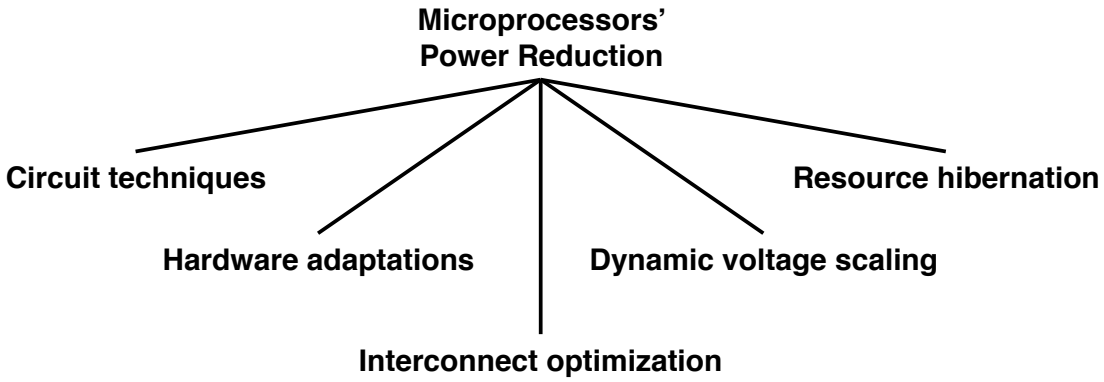


Figure 1.4: Techniques used for reducing the power consumption of microprocessors.

consumption since used as the medium of most electrical activity. Hence, **interconnect optimization** is also employed and focused on improving buses or sharing buses among multiple functional units.

Among dynamic consumption related techniques, **dynamic voltage scaling** (DVS) addresses the problem of adapting the processor's clock frequency and supply voltage to the variations of the executing workload. *Dynamic voltage and frequency scaling* (DVFS) is a common technique for the reduction of CPU power consumption. It is based on scaling voltage and frequency of the CPU. Because of the complex nature of workloads, this is mainly a hardware additional functionality to be exploited by upper software layers (see Section 1.2.3).

The reduction or zeroing of the energy consumption during the idle periods is another key point for improving the efficiency of microprocessors. **Resource hibernation** techniques power down components (disks, network interfaces, displays) during idle periods.

Storage systems are also accounted for improving the energy efficiency of data centers,

since estimated to be responsible for about 25 to 35% of the total power consumption [13]. Consumption of hard disks can be reduced operating on disk's spin and seek [33], and on the control power sub-systems [2]. As for the **memories**, two main approaches are adopted for limiting electricity consumption: (i) reduction of energy required for memory accesses and (ii) reduction of the number of memory accesses. By splitting memories in multiple banks, energy consumption is reduced since each bank can independently transit into different power modes or only needed circuits are activated [52]. Specialized cache structures, which dissipate less energy, can be used for reducing the access to the memory [75]. Another approach is to use *trace caches*, which, although developed for performance, also showed power benefits [65].

Adopted **network devices** also affect the total energy consumption of data centers. In [20], the authors create a general model of the power consumption of various configurations of routers and study the potential impact of power-awareness in a set of example networks. Results show that there are many opportunities for network efficiency up to reducing electricity consumption of an order of magnitude. Different technologies have been proposed in this context, both for reducing the power absorption and for improving performance. Pure optical switching architectures can provide terabits of bandwidth at much lower power dissipation than current electronic based devices [7]. Also for the subclass of network devices, dynamic power management strategies are used for reducing the energy dissipation of the components according to the network traffic [59]. Authors of [85] develop a technique to reduce the activities of network processors in accordance with the traffic volume. By monitoring the average number of idle threads in a time window, some processors are turned

off when unnecessary. This also requires redirecting some network packets, determining the thresholds of turning on/off processors, and avoiding packet loss. In [140] a high performance router at low power consumption is presented. It is based on the integration of ASICs/FPGAs and memories and on a scalable architecture. Packet classification is discussed in [74, 73]. In [74], the proposed router architecture consists of a clocking unit that dynamically changes the clock speed to match traffic fluctuations. Classification algorithms and hardware acceleration are used in [73].

Much work has been done for improving the hardware aiming at *greening* data centers. Some of the discussed technologies can contribute to the energy consumption reduction in the overall system thanks to improved hardware design. This helps both in dynamic and static aspects of a whole data center, even though actually neglecting *what the system does*. In other cases, dynamic power management is adopted for hardware components. However, while meaningful in the design/building phase of a processing system, in the case of an existing data center, other approaches should be considered.

Cooling

The cooling subsystem of data centers has a large impact on the total energy consumption. While the power required to remove the heat dissipated by the components in a rack is about 10% of the power consumed by the components, the power required to remove the

heat dissipated by all the racks in the data center is about 50% of the total energy consumption [104]. This percentage may also increase due to the formation of hot spots [116]. Efficiency of cooling room air conditioning (CRAC) is usually measured with the *Coefficient of Performance* (COP), which is defined as the ratio between the amount of heat removed by the cooling system and the total energy it requires (see Section 1.3). Several approaches have been proposed for increasing the COP of cooling systems.

Authors of paper [80] propose an approach that optimizes both the air conditioner compressor duty cycle and the fan speed for reducing the difference between the heat generated and extracted from a machine, hence minimizing the cost of cooling and the risk of hardware damage. In [38], authors discuss the importance of the design of data centers in order to improve the energy efficiency of the system. The paper adopts a simulation-based approach to identify the design that brings adaptability and robustness in the multi-scale convective systems. Similarly, patent [119] describes the design of a data center and cooling system to reduce noise and energy costs while increasing usable space. Another approach for reducing the energy consumption of the cooling system is based on the adoption of liquid in place of air [77].

1.2.2 Software

Software running in a system can significantly impact on the overall energy consumption. Also, the efficiency improvement that can be achieved with *green software* can be more important than the one achievable with the micro-management of resources at the hardware

level [121]. Developers can use different techniques for producing energy efficient software, which reduces the consumption of drivers and applications.

Authors of [21] highlight the importance of facing energy related issues in the early stages of software development. They propose to modify the **software development life cycle** by considering energy efficiency as a non-functional requirement already in the software requirements specification. The importance of the development process for green software is also discussed in [37]. The authors present a method for producing green software with agile methods. The driving idea is to use the continuous integration approach and to instrument the source code on the test classes to start and stop measurement of energy and performance data and to add the measured data to the test results so to continuously monitor, and possibly reduce, the consumption of tested software. The approach proposed in [125] is based on a similar idea but only focused on the design phase. Authors use software architecture graphs (SAGs), which capture how the software has been built out of processes and events. The base energy consumption and variations due to SAG's transformations are estimated by means of simulation.

In [110] authors discuss about the ***race to idle***, since assuming that by improving performance saves not only time but also energy. The idea is that the faster the workload is completed, the earlier the computer goes back to idle and the more energy is saved. For improving performance, efficient algorithms, multithreading and vectorization are introduced. The importance of the idle state is also debated in [111], but with a different meaning. Authors observe that a non-energy efficient application, apart from not using the minimal energy, can also prevent all the power management features build into the system from

properly working. Possible approaches for reducing the consumption of idle applications are the reduction of the interrupt rate, the use of cache locality and efficient multithreading.

Compilers also can significantly contribute to reduce the consumption of software, although limited to a specific application and to a specific processor [131]. They can optimize performance by reducing the execution time so also saving energy. Other optimizations can be achieved by eliminating redundant load and store operations so to reduce the energy dissipated during memory accesses. For mobile devices, remote compilation is employed for demanding compilation and execution of applications to remote and more powerful servers [103]. Dynamic compilation addresses some of the problems related to the *una tantum* compilation, which makes applications optimized without knowing its actual use. Programs are monitored during execution and recompiled if there are changes in the runtime environment (e.g., resource levels) [76].

If the target system provides computational resources to final users that run their own applications, these approaches are not useful. These users are not likely to be interested in reducing the consumption of the infrastructure hosting their applications or running their jobs. From the service provider point of view, s/he can not modify user applications running in the system, but certainly s/he looks for techniques that reduce the consumption by better exploiting existing hardware for the given workload. In other words, both hardware and software techniques are useful for new generation, private processing systems, which are build with hardware exploiting new technologies and for executing own applications that can be *greened* as needed. For an existing system, dynamic power management can make

it energy-proportional, which means the power consumed is in proportion to the amount of work performed.

1.2.3 Management

Approaches discussed in this Section try to better exploit available resources for a given workload. Although related to dynamic power management, in this category we do not take into account dynamic approaches adopted for some hardware devices already discussed in Section 1.2.1. Turning on/off machines, power-aware scheduling or consolidation algorithms, and green load balancing are commonly exploited by management techniques for improving energy efficiency.

Since electricity production is a main cause for pollution, green energy is also evaluated in some works. As an instance, photovoltaic solar arrays are discussed in [56]. In the design phase, also the location where a data center is implemented is a key decision. Reducing the temperature has a cost in terms of energy consumption (due to the work of the cooling sub-system). This can be achieved by installing the system in a cold place.

Scheduling and Load Balancing

In the case of load balancing and scheduling algorithms, the main idea is to equally distribute computation and communication load amongst the processors (both before the computation start and after by means of migration) aiming at improving some aspects. Here we focus on consumption and performance aspects, that is, on those approaches that either reduce the

energy consumption without affecting the performance or improve the performance without affecting the consumption.

Improvement of electricity consumption by exploiting **green energy** is discussed in [56]. The authors propose a parallel batch job scheduler for a data center powered by a photovoltaic solar array; the electrical grid is considered as a backup. The scheduler increases green energy consumption up to 117% and decreases costs of 39% by reducing workload scheduling when few solar energy is estimated to be available in the near future. When deadline violations can not be tolerated, the electrical grid is used, but the scheduler selects times when it is cheap. Results on a 16-nodes cluster also demonstrate the capability of the scheduler to not affect performance.

The importance of the communication time between processors is debated in [22]. Authors assume a number of tasks to be executed in a distributed system and divide them into many subtasks that can be concurrently executed on different processors. The workflow among tasks is represented with a Directed Acyclic Graph (DAG). The goal of the scheduling is, obviously, to minimize the finish time of DAG. At this aim, full parallelism is used. This does not always corresponds to less executing time. The proposed heuristic power-aware scheduling algorithm, not only minimizes the finish time, but also produces more slack time and provides maximum slowing down potentials for energy reduction. Experimental results show the algorithm being able to reduce energy consumption on average by 36%.

Dynamic frequency and voltage scaling is also exploited by upper layers of system architecture components. In 1996, it was released the first version of the **Advanced Configuration and Power Interface** (ACPI) specification, which provides functionalities to control the power management of the hardware devices. The operating-system-level power management (OSPM) was introduced providing to the OS and other upper-layer applications a general interface to address the configuration and power information of individual devices or of the whole computer system. In [50], the authors apply DVFS on high performance computing (HPC) for reducing power consumption. Results show that DVFS-based approaches help in significantly reducing the energy consumption of cluster systems (up to 36% savings) while not affecting the performance. Authors discuss the variation of energy saving with the compositions of various workloads, application types, and system configurations.

Since the cooling sub-system significantly impacts on consumption and, in turn, its work is due to temperature, techniques for **reducing the temperature** are also assessed. The frequency of the CPU affects the temperature of the CPU itself and of the whole system. A common approach is to reduce such a frequency if a certain temperature is reached [116]. The authors discuss a load balancer aiming at reducing the power consumption due to parallel applications through temperature control for cooling power reduction. Exploiting the DVFS, CPUs are allowed to work at the maximum frequency until a threshold temperature is reached. When the frequency of all the cores is reduced, the input voltage reduces as well, resulting in power saving. The temperature threshold is fixed on average temperature. Exploiting frequency levels of modern processors, when the temperature exceeds the threshold, the frequency is reduced by one level. The approach has been applied to a 40-nodes

testbed data center with a dedicated CRAC showing a timing penalty in the range of 2-20% in face of a power reduction of up to 57%.

Models can be used for **predicting trends** with respect to newly submitted workload and then selecting the best schedule. The machine learning approach is used in [9]. The framework both considers power consumption and workload features, onto which it applies a variety of techniques covering the whole control cycle of a real scenario. When a job arrives, the system allocates it on a host by adopting a common scheduling approach, then uses the model to figure out if there is a better allocation. In such a case, the job is moved form a host to another one. The new allocation is selected on the basis of the expected values of a performance metric R and of a consumption metric C. As a performance metric, the probability of completing the job in a time period is adopted. This evaluated the chance that the user requirements are satisfied. As for the consumption, power dissipation is used. Some doubts arise for the selection of the performance metric, which is neither real performance (how much the system produces) nor dependability (SLA violations is too generic as a dependability metric since not all violations correspond to failures). The evaluation of the approach is done through the simulation of a 400-nodes data center executing a heterogeneous workload. The SLA fulfillment is worse by about 1%, but the overall power consumption is better by 10%. Paper [142] examines three cluster allocation policies, two of them aiming at improving performance and the third one optimized for energy conservation. Authors assume two types of processors in the cluster, each with its own performance and energy characteristics, and job service times are considered unknown to the scheduler.

A model is used to evaluate the performance and energy behavior of the policies. Simulation results indicate that each of the three introduced policies has its own advantages and disadvantages. The SQEE energy aware policy reduces the energy consumption of the system without significantly affecting the average job response times. The SQHP policy (optimized for performance) outperforms the other policies but electricity consumption also increases. The PBP-SQ policy presents as the worst, especially at high load, in improving the energy consumption. Interestingly, the results empirically show that the selection of the management policy mainly depends on the system load and on the trade-off one want to achieve between performance and energy consumption.

Another approach for identifying the best balancing of the workload is to formalize **optimization problems**. In [83], two combinatorial optimization problems are defined: minimizing the schedule length with energy consumption constraint and minimizing the energy consumption with schedule length constraint. Schedule length minimization is a typical problem of multiprocessor and multicore processor computing systems, where energy consumption is an important concern, and in mobile computers, where energy conservation is a main concern. Schedule length constraints are typical of real-time multiprocessing systems, instead. The scheduling problems are defined for the optimization of the energy-delay product. For each problem one factor is fixed and the other is to be minimized: the schedule length is minimized by consuming a given amount of energy or the energy consumed is minimized without missing a given deadline. The attention is on parallel tasks on multiprocessor computers with dynamic voltage and speed. To problem is divided in three sub-problems to reduce the complexity: system partitioning, task scheduling, and

power supplying. Achieved results are experimentally validated. A multiprocessor computer is divided into clusters of equal sizes and tasks of similar sizes are scheduled together to increase processor utilization. It is found that the proposed heuristic algorithms are able to produce solutions very close to the optimum.

Given the impact on consumption of the cooling system, **thermal aware workload placement** is also considered as a possible approach for reducing energy consumption [136]. For describing performance and temperature, an analytical model is used (although the discussion in the paper is not able to explain the actual model, just bases for the modeling are reported). Two scheduling algorithms are then evaluated: Thermal Aware Scheduling Algorithm without (TASA) and with Backfilling (TASA-B). The algorithms are evaluated through simulation showing that TASA can reduce the average temperature by 16.1°F and the maximum one by 6.1°F, with a performance decrease by 13.9%. TASA-B can reduce the average temperature by 14.6°F and the maximum one by 4.9°F, with a performance degradation by 11%.

Cloud Computing and Virtualization

The issue of energy efficient resource allocation is studied also for cloud environments. Given the large diffusion of cloud computing and its adoption in data centers, scientific researchers are studying how to exploit such a technology also for improving the consumption of computing systems. Usually, an Infrastructure as a Service (IaaS) cloud is used for delivering basic storage and compute capabilities to handle the possible workloads [97].

The placement problem *of* virtual machines (VMs) on physical machines (PMs) is known in scientific literature as **VM consolidation problem**, since studying how to consolidate the VMs on a small number of PMs in order to reduce energy consumption. To find an energy efficient distribution of the load among running machines, two main observations are to be done: (i) a reduced load for many machines requires a huge static consumption (due to many switched on machines) and, on the other hand, (ii) if only few machines run all the workload, their consumption is large and performance may be affected.

The placement of tasks *on* VMs is also debated in many works, similarly to the common scheduling or load balancing problem.

Figure 1.5 shows two possible approaches for the consolidation of VMs. In the case (a), a PM Server A has one active VM that is migrated to Server B, so that Server A can be switched off. This may compromise performance on Server B and increase the execution time (which may worsen the total consumption for executing tasks), due to the dependencies between energy consumption, resource utilization, and performance of consolidated VMs, which share and compete for the resources of hosting servers (such as CPU, main memory, and I/O) [57]. Hence, the case (b) can be also considered, where the load is balanced between the two servers, each with two running VMs.

In [81], the authors study the trade-off between performance and resource utilization when multiple VMs are hosted together, and the effect of different VM's resource requirements (in terms of CPU, network and storage). The specific behavior is dependent on the

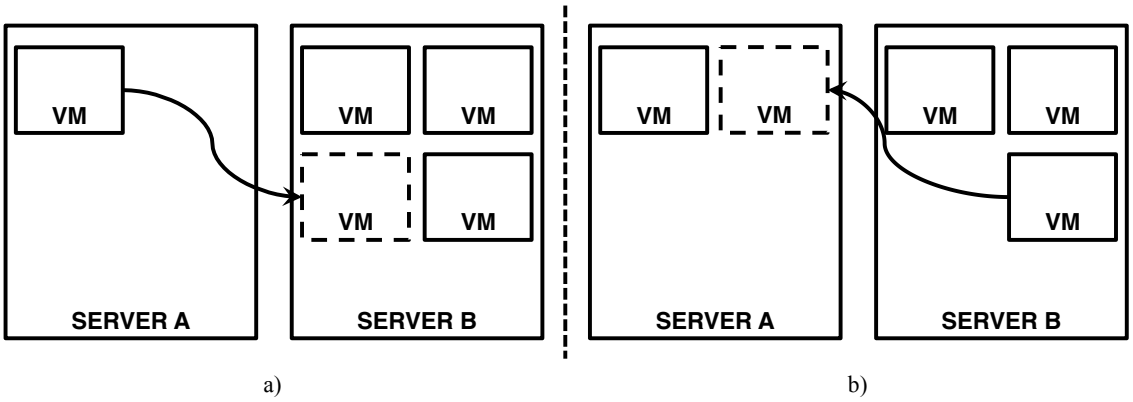


Figure 1.5: Possible approaches for the consolidation of VMs.

type of resource and on the type of the workload. For compute intensive tasks, performance degradation gradually decreases when the number of co-hosted VMs grows. But, in the case of large memory footprint, the degradation is faster. In the case of I/O operations, performance degradation occurs due to additional delay for data processing. These results demonstrate that the consolidation problem must be solved on multiple dimensions. CPU is not the only critical resource, but also memory, storage and network have to be considered. Similar observations will be made for our case study in Chapter 6.

In [86] ecoCloud is presented as an approach for workload consolidation that efficiently exploits resources through the balancing of CPU-bound and memory-bound VMs. The placement is performed by taking into account the utilization level of each server, aiming at consolidating the workload on as few servers as possible. A server that is over-utilized or under-utilized on either CPU or memory is not used. In fact, in the case of over-utilization, the reduction of the quality of service is avoided. In the case of under-utilization, the objective is to allow the server terminating its operations as soon as possible so that it can

be switched off. A server with intermediate utilization is the one on which the new VM is placed. The approach has been experimented on a real data center with 28 servers running both CPU-bound and RAM-bound VMs. It reduces power consumption while limiting the number of VM migrations and server switches and balancing CPU-bound and memory-bound applications. In [57], it is proposed an approach for consolidating VMs in an IaaS cloud. The approach is based on the use of DRBD, a module for high-availability data storage in a distributed system, which provides the storage abstraction independently from the running VM. That is, the VM can be easily migrated from a server to another server avoiding to move also large quantity of data. The DRBD module takes care of replicating data for improving the availability. In this way the overhead for multiple disk operations on the same server is reduced and demanded to network interfaces. The approach has been implemented for a cloud platform and results show that, during normal execution phases, performance is quite unchanged while energy consumption is reduced. In [133], authors compare consolidation strategies on the basis of their impact on performance, energy, and reliability aspects. The problem is faced with multi-objective optimization, by developing an utility model that unifies multiple constraints on performance SLAs, reliability factors, and energy costs. The global optimal solution is found by means of a genetic algorithm exploiting a collection of reliability-aware resource buffering and VM migration for generating good initial solutions and improving the convergence rate. Simulations show that the approach can be used to find good reliability-consumption and performance-consumption trade-offs. However, employing "SLA violations" as a reliability metric is not always correct. As in this case, where it simply represents the performance going below a certain threshold.

In [137], the authors discuss the difficulties in characterizing the network bandwidth demands of virtual machines, since it is dynamic. The VM consolidation problem is formulated as a Stochastic Bin Packing problem and an online packing algorithm is proposed. Commonly, the consolidation problem is represented as a Bin Packing problem, solved with the First Fit Decreasing (FFD) bin packing heuristic, which packs the next VM to be provisioned on the first server it fits. On the contrary, the Stochastic Bin Packing assumes requested resources aleatory; hence, the items to be packed are a set of independent random variables. Numerical experiments demonstrate that the proposed algorithm can save up to 30% of servers without violating the server capacity constraints. In Chapter 6 we adopt the FFD bin packing heuristic, considering clusters of loads whose resource request can be assumed deterministic.

As for task **scheduling *on* VMs**, several categories of techniques commonly adopted for physical servers are compared with respect to their impact on energy efficiency in [92]: (a) first-come first-serve with back-filling (FCFS-backfill); (b) earliest deadline first (EDF); and (c) an offline genetic algorithm (SCINT). Simulation results show that the thermal-aware enhancement achieved with FCFS-backfill is the smallest and depends on the intensity of the incoming workload, while the genetic algorithm is able to improve energy saving up to 60% with respect to FCFS. The performance of EDF is similar to the one of SCINT for low loads but it degrades to the performance of FCFS-backfill for high loads. EDF is also significantly faster than SCINT in scheduling jobs.

In [82], the authors show how to reduce the energy consumption of virtualized computing systems avoiding the under-utilization of resources and idle power consumption. The problem is formulated considering the consolidation of n tasks on r cloud resources without violating time constraints. Two consolidation algorithms are proposed, each with a different objective, namely maximization of resource utilization or minimization of energy consumption. Results achieved with simulation show that energy consumption can be reduced up to 18% with respect to random scheduling algorithms.

Tools

Several tools have been developed for evaluating the energy efficiency of processing systems. In 2007, the SPEC organization introduced SPECpower_ssj2008 [122], "the first industry-standard benchmark that evaluates the power and performance characteristics of single server and multi-node servers". The benchmark both measures power and a performance metric in order to figure out their relations and allow the analysis of the energy efficiency of data centers. Adopted workloads for performance evaluation exercise CPUs, memory hierarchy, the Java Virtual Machine (the workload is written in Java programming language and the unit of measurement for performance is *Java operations per Watt* or *Server Side Java Operations per Joule*), and some aspects of the operating system. As for previous SPEC's benchmarks, the organization also provides some guidelines (see Section 3.2.1).

GBench [123] is a benchmark explicitly proposed to evaluate the energy efficiency of HPC systems at different workloads. The proposed methodology is applied to scientific

data centers to identify application parameters impacting both performance and energy consumption. Number of processes and block size (HPL parameters) appear to be the more affecting ones. Also, authors show that there is a correlation between power and performance related activity (e.g., cache misses and power consumption).

Ge et al. in [51] propose PowerPack, a framework for evaluating the power consumption of devices in high performance clusters such as processors (also multicore and multiprocessor nodes), disks, memory, NICs. The framework also correlates these measurements to parallel applications running in the system. It is used to study the power dynamics and energy efficiency of DVFS techniques showing their capability in enhancing system energy efficiency while maintaining performance. In [46] authors present a framework for the profiling of power consumption of scientific applications on HPC systems. The framework is used to study the energy efficiency of a 32 nodes cluster.

It is worth noting that these benchmarks do not take into account the erratic behavior of the system, but just the correct execution of the provided benchmark programs. An effective benchmarking should perform measurements for normal data center practice and workload. They are useful to evaluate the energy efficiency characteristics of the adopted hardware, or to identify management techniques that can be used to reduce the consumption or to increase the performance in some conditions, but they do not evaluate the failures that can happen in the system. It would be useful a tool that brings out the behavior of the system also in presence of failures. For instance, this can be achieved by merging the actions of the GBench [123] tool with the ones of DBench [72], a dependability benchmark,

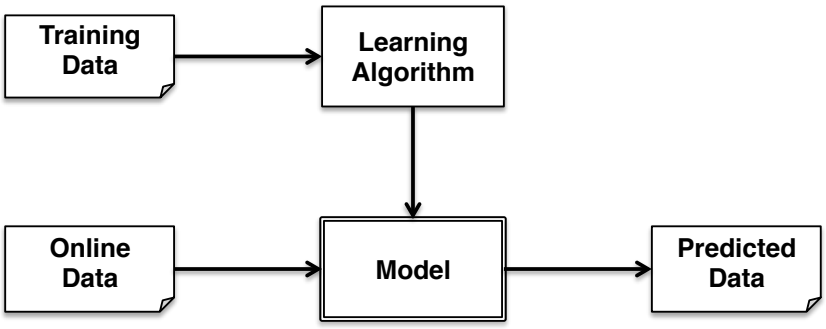


Figure 1.6: Common model based approach for estimating future behavior and tuning the system.

and by monitoring the occurrence of failures and related power consumption.

Models

Some of the discussed papers are based on models for the estimation of performance and consumption. Such models can be used during the design and development of a system to determine the number of machines to be used, the scheduling and the management policy required to offer a specific performance degree, and to estimate the related energy consumption. In the operational stages, models can be used by tools for the online estimation of the system performance and consumption and for tuning parameters of the dynamic management techniques to maintain a certain performance or consumption level.

In [9], a machine learning approach is used for creating models that are then populated with online data to estimate the future behavior and make decisions. Components and steps of the model based approach are depicted in Figure 1.6. Some data are used as input for a learning algorithm (of whatever kind, e.g., machine learning, flow intensity invariants

[115]), which produces a model. Models of application and machine behaviors learned from previous system trends are used to predict power consumption levels, CPU loads and SLA timings. These are helpful in the selection of the changes/actions to be done on the system for maintaining QoS while reducing energy consumption. For model creations, authors use computationally light predictor algorithms. The goodness of the predictors is checked against the correct values on the test set.

Ghosh et al. [54] conducted a joint study of performance and power consumption by means of stochastic models. Authors propose a performance-consumption analytic model for Cloud systems, and provide an approach to be used to configure the data center taking into account power-performance trade-offs. The model is a Markov reward model created with an interacting sub-models approach for reducing complexity and chance of error of a single monolithic model. To make all the sub-models homogeneous continuous time Markov chains (CTMC), all inter-event times are assumed to be exponentially distributed. In [53] a cloud availability model is proposed. For backup and recovery, as well as for reducing power consumption, PMs are assumed organized in three pools: *hot* (running PMs), *warm* (turned on, but not ready PMs), and *cold* (turned off PMs). When using default VM images, the deployment on hot PMs can be performed with minimum provisioning delay. The deployment on a warm PM requires additional provisioning time (to make the PM ready). Further delay is added when using PMs in the cold pool (which need to be turned on before being used). Different modeling approaches are considered. Apart from the common monolithic modeling, an interacting sub-models approach is proposed to solve the largeness problem [112]. The simulation is also used to avoid the generation of the large state-space

of the Markov chain. Results show that the monolithic model becomes intractable and fails to produce results as the size of the Cloud system increases and demonstrate the scalability of the interacting stochastic sub-models approach and of the simulation.

In [142], the processing system is modeled as an open queueing network to compare three scheduling policies. Simulation results show that the system behavior mainly depends on the load more than on the policy. Also, the results support the selection of the policy to achieve the desired trade-off between performance and energy consumption.

1.3 Metrics

Metrics used in discussed papers aim at measuring *how much green a data center is.* This is done for example, by calculating the consumption or estimated gas emission per time unit, by comparison with similar data centers, through the relation between performance and spent electricity, by assessing thermal effects. These metrics can be classified as shown in Figure 1.7.

Gas emissions are often contemplated since one of the main causes for the attention on the energy consumption of processing systems. Fine-grained measurement of such emissions is not easy due to the large number of factors that should be contemplated (e.g., running applications, network packets). A coarse estimation of gas emission of a data center can be performed on the basis of the total energy consumption and emission to generate one unit of power (kWh) [136].

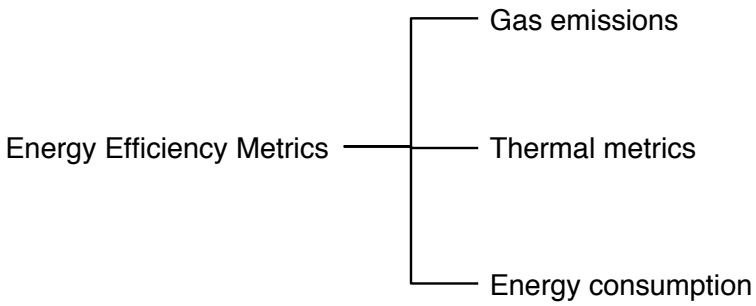


Figure 1.7: Classification of energy efficiency metrics.

Thermal metrics are used given the large impact of the cooling system on total data center energy consumption. Furthermore, temperature also affects the chance of failure [40]. For improved reliability, the temperature of the server farm should be in the range 20-24°C. The efficiency of the cooling sub-system (cooling room air conditioning - CRAC) is also evaluated. A common metric is the *Coefficient of Performance* (COP) relating the amount of heat removed by the cooling system and the total required energy. It can be demonstrated that the total power consumption of a data center has two components, one due to the working nodes of the system (P_{wn}), and another one due to energy consumed for cooling such nodes, which depends on the COP, as in the following equation [102]:

$$P_{TOT} = P_{wn} + \frac{P_{wn}}{COP} \tag{1.1}$$

The *Airflow Efficiency* (AE) measures the total power required by fans per unit of airflow. It provides an overall measure of how efficiently air is moved through the data center. The unit of measurement is W/cfm (watts per cubic feet per minute) and 0.6 is a standard value [87]. The *Rack Cooling Index* (RCI) is used for the efficiency of the cooling system of

the racks used in a data center. Commonly, the passings of upper and lower temperature thresholds are counted. *Return Temperature Index* (RTI) and *Recirculation Index* (RI) consider the relation between inlet and outlet temperature.

Energy consumption metrics are based on the total consumption of a data center. This is then related to some other metrics. The Performance/Watt metrics are a *de facto* standard in measuring the energy efficiency of processing systems. An example is the use of *FLOPS per Watt* [25].

The Standard Performance Evaluation Corporation (SPEC) provides Java benchmark programs for the evaluation of the energy efficiency; the number of Java operations performed per Watt consumed is adopted as a metric [122].

The *Data Center Infrastructure Efficiency* (DCiE) is defined as [31]:

$$DCiE = \frac{\text{IT Equipment Power}}{\text{Total Facility Power}} \tag{1.2}$$

where "IT Equipment Power" represents the consumption of the IT equipment (e.g., servers, storage, network devices, monitors), while "Total Facility Power" also includes devices supporting the IT equipment, such as the cooling system. The reciprocal, *Power Usage Effectiveness* (PUE) [31], is used to evaluate how much energy is required for each unit used by the IT equipment.

The Heating, Ventilation, and Air Conditioning *(HVAC) system effectiveness* [87] is the fraction of the IT equipment energy used by the HVAC system. It is the ratio between the total IT consumption and the sum of the electrical energy for cooling, fan movement, and

any other HVAC energy use.

The *SWaP* metric (Space, Watts and Performance) [90] estimates the energy efficiency of a data center as performance related to occupied space (in terms of rack units) and consumption (watts):

$$SWaP = \frac{\text{Performance}}{\text{Space} \times \text{Consumption}} \tag{1.3}$$

The *Data Center energy Productivity* (DCeP) metric [31] is the one more similar to the efficiency commonly adopted in physics, relating the useful produced work with the total energy consumption of the data center for producing that work. The useful work is quantified as the tasks performed by the hardware within an assessment window. As it is shown in Chapter 5, this throughput is not always capable of representing the actual useful work performed by a system.

1.4 Remarks

The importance of evaluating the energy efficiency of processing systems is evident. They contribute to pollution in a non-negligible way. Also, the operational cost due to electricity is now huge. Technology is all the way and should help in producing green systems. However, this is not enough and, above all, this is not useful for existing data centers. It is not always possible to change the hardware of an exiting processing system, its cooling system, and even less its geographical placement for reducing electricity costs and thermal issues. Software can be *greened* too, but for data centers providing services to third parties it is not possible to change the actual load of the system. For these reasons, management techniques

are often seen as the most feasible solution for improving the energy efficiency of a system. Different metrics exist for estimating this efficiency. Most of them relate a performance metric (e.g., operations per second, mean job duration) with a green metric (e.g., energy, power, temperature). Nevertheless, two main concerns arise for this idea of energy efficiency for processing systems:

1. *is this really energy efficiency?* Commonly, in engineering and physics, energy efficiency measures the relation between the system's useful power output and the provided power input. In thermodynamics, different metrics are adopted relating performance and input power, but the amount of useful work output is a function of the amount of high-temperature heat input. The characteristic of all such metrics is to relate what it is furnished to the system to its capability of turning this into useful work. Hence, what is to be considered is *what the system wastes.*

2. *Why failures are not contemplated?* It is known that large scale processing systems experience failures, which happen more frequently as the size (i.e., number of components and interconnections) of the system increases. The energy consumption due to this failures seems to be a good candidate for the estimation of the squandered input power of the system. In Chapter 5 it is shown that failures can be accounted for the 9% of the energy consumption of a batch processing system with 500 nodes. Also, we show that performance may be not representative of the malfunction, above all in not saturated systems, which are the majority of real cases [109, 132, 113]. If not properly assessed, the cost of the failures remains hidden together with dependability issues.

In the next chapters of this dissertation it is shown that ***performance***, energy ***consumption***, and ***dependability*** represent the three main aspects of processing systems, and they are strictly related.

Chapter 2

Performance, Consumption and Dependability

Performance aspects have always been seen as meaningful for the effectiveness of processing systems. Then, also dependability aspects started to be considered given the common failure prone behavior. When fault tolerance was introduced, and systems could work also in a degraded mode, the concept of performability was introduced as the capacity of a system of working well also in presence of failures. Nowadays, with cost and pollution related problems, the consumption is recognized as peculiar for the evaluation of data centers. Energy efficiency relates performance to consumption, quantifying how much energy a system consumes to produce a certain quantity of work. In this chapter, the basic definitions for the three aspects of performance, dependability, and consumption are given and their relations are discussed.

2.1 Basic Definitions

Discussion on energy efficiency of the previous Chapter 1 highlighted that performance and consumption are two main aspects of computer systems. Most of the scientific literature is also about dependability aspects of such systems. There is a relation between dependability and performance and, as it will be showed in this dissertation, between dependability and consumption. Such relations are outlined with a set metaphor presented in Figure 2.1. The three aspects are presented as circles, each representing a set. Some studies are related to a specific aspect. The relation between dependability and performance is referred to

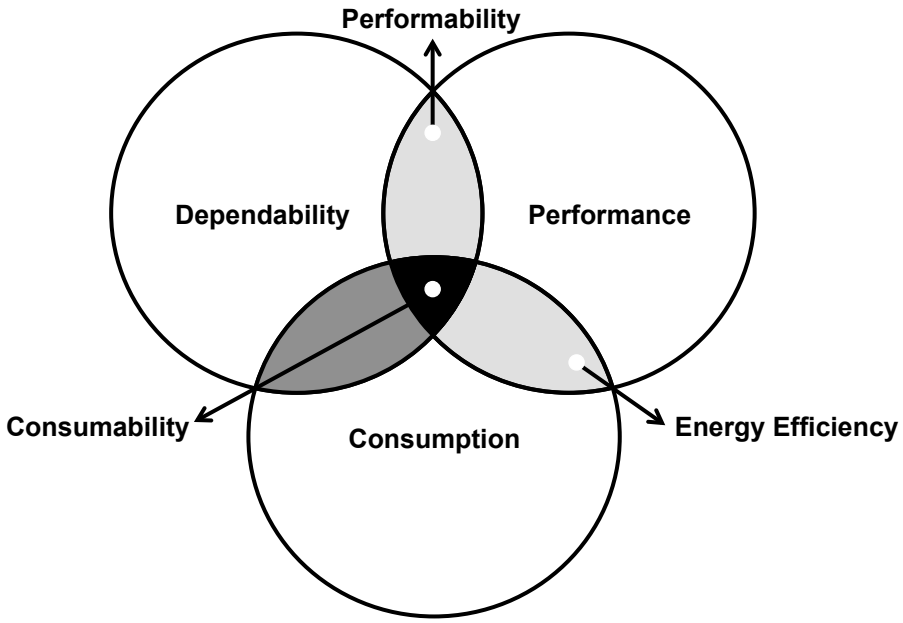


Figure 2.1: A set view of consumption, performance and dependability aspects for processing systems.

Table 2.1: References defining the main aspects of processing systems and their intersections.

Aspect	**References**
Performance	[84, 94, 100, 3, 78]
Energy consumption	[10]
Dependability	[5, 68, 48]
Performability (Dependability-Performance)	[8, 89, 48, 135]
Energy efficiency (Performance-Energy consumption)	*see Chapter 1*
Consumability (Consumption-Dependability-Performance)	[40, 88, 32, 39, 117, 36]

as Performability [89]. The energy efficiency is the intersection between performance and consumption aspects. Other relations are mainly unexplored. In this section, we discuss the basic concepts, while their relations are explored in the next ones. The introductive and exemplifying references for each set are reported in Table 2.1.

A **processing system** is here a set of interacting components (e.g. worker nodes, interconnecting network, power supply units, cooling sub-system). It can in turn interact with other systems, softwares, or humans. A processing system has a certain **function** representing what it is intended to do. A system produces **useful work** when performing what described by the functional specification [5]. The **service** delivered by a system is what the system actually does to implement its function. It can be represented as a sequence of states traversed by the system as perceived by the user. Correct states are the ones in which the system accomplish its function. In the case of a **batch processing systems**, which elaborates jobs submitted by final users without interaction apart from the submission procedure, this function corresponds to the returning of correct results in a certain time. As for the useful work, we consider it being produced when a job is completed in the shortest time and traversing only statuses in which it is expected to be (that is no failure/repair events happen). The **administrator** is a person (or group of people) responsible during the design, testing and operational stages of the system life cycle, has the authority to manage, modify, repair and use the system; and also cares for costs of the system. Final users, or simply **users**, are the entities (humans or other systems) that receive the service from the system.

2.1.1 Performance

The concept of performance is as abstract as the term is common. Commonly, the performance of a processing system is meant as the *quantity* of useful work produced.

Performance analysis should be thought of as a combination of measurement, interpretation, and communication of a computer system's *capacity* [84]. The main issue is to figure out the capacity one want to measure. This may vary depending on the situations. In some cases, performance analysis can be used to compare several alternatives for selecting the hardware for a processing system, or to evaluate the impact of adding additional resources to an existing one. In other cases, the goal could be the tuning of the system in order to adapt it to the common load. As a consequence, the first step is the selection of the parameters to be evaluated and of the related metrics. Typically, what is measured is the *count* of how many times an event occurs, the *duration* of some time interval, or the *size* of some parameters. Event counts can be normalized to a common time basis, so achieving a speed metric called *throughput*.

One of the main indications of performance of a computer system is the **clock rate**, i.e. the frequency of the processor central clock. Given the commonly large values achieved nowadays, the **number of CPUs** is also frequently used for large computer installations. Other performance metrics count the millions instructions executed per second (**MIPS**) and the millions of floating-point operations executed per second (**MFLOPS**). Considering programs that computer systems execute, the **execution time** a computer system requires

to execute a program is often adopted as a performance metric. In 1988 the Standard Performance Evaluation Corporation (**SPEC**) was founded with the aim of "produce, establish, maintain and endorse a standardized set" of performance benchmarks for computers [122]. The organization provides a set of benchmark programs representative of the common computer systems workloads. It also standardizes the methodology for measuring and reporting the performance obtained when executing those programs. This allows different systems to be compared. Other metrics are not related only to the processor characteristics. For large scale computing systems, executing a large number of batch jobs, other metrics are adopted. The **response time** is adopted as the time that elapses from the submission of a job execution request to the system until the return of the result from the system. The **throughput** can be measured in terms of the number of jobs completed in the unit of time [94, 100]. If network characteristics are relevant, the **bandwidth** is a measure of the number of bits that can be transmitted across the network per second. More recent metrics are related to the *number of traversed edges per second* [3] or *performance vectors* [78].

2.1.2 Consumption

Electricity consumption is commonly measured as **watts per hour** ($W \cdot h$). The watt measures the power, i.e. the rate of energy conversion or transfer (amount of energy consumed per unit time). Hence, the power is the rate at which energy is generated or consumed and is measured in units (e.g. watts) that represent energy consumed per unit time. The *joule* is the unit of measurement for energy (International System of Units). It is equal to the

energy expended (or work done) in applying a force of one newton through a distance of one meter. Watts per hour can be easily transformed in *joule*s by using seconds in place of hours ($1\ W \cdot h = 3600\ J$).

Consumption of processing elements is modeled in several works. An example related to microprocessors, but easily extensible to set of components or entire systems, is given in [10]. The total power consumption is expressed as:

$$P_{total} = (\sum_{i=1}^{c} ompsAR_i \times P_i) + P_{static} \tag{2.1}$$

where P_i is the weight of component i and AR_i is its activity ratio. The $AR_i \times P_i$ represents the dynamic power consumption of component i, and P_{static} represents the overall static power consumption of all components. For large processing systems, a very similar consumption model can be achieved (see Section 4.2.2).

It is worth noting that in this dissertation we use the expressions *"reducing energy consumption"*, *"reducing power consumption"*, *"reducing power dissipation"* and so on, almost interchangeably, even though optimizing the power does not mean optimizing the energy. Energy can be intended as the accumulation of power over a period of time; hence, a large decrease of the instantaneous power dissipation may correspond to a significant increase of the total computation time and of the energy consumption. It is detailed when referring to a specific physical concept.

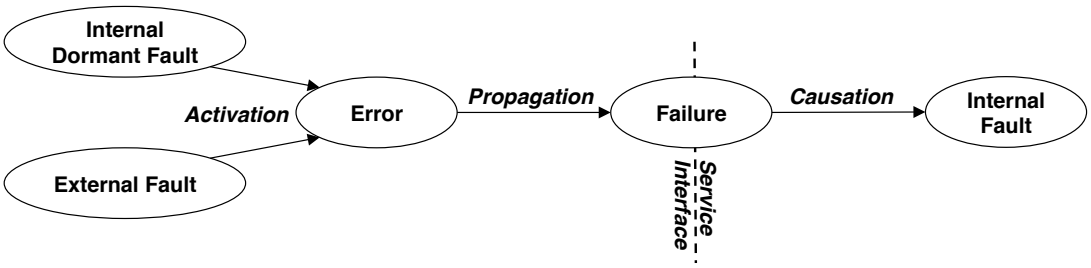


Figure 2.2: The fault-error-failure chain.

2.1.3 Dependability

In this dissertation, it is considered the dependability of processing systems as defined in [5]: the ability of a system to deliver a service that can justifiably be trusted. It is an integrated concept encompassing the attributes of reliability, availability, maintainability, safety, and integrity.

The Fault-Error-Failure Chain

When a computer system is not able to properly deliver the correct service, a **failure** is said to be occurred. Considering the correct service as a sequence of system states, a failure occurrence means that the system went at least in a state that deviates from the correct ones. The deviation is called **error**. The cause of an error is called **fault**. Referring to the well-known and widely accepted taxonomy presented in [5], faults, errors and failures are related as shown in Figure 2.2.

A **dormant fault** produces no effects. For producing an error, a fault is to be activated and is named **active fault**. A failure occurs when the production of an error propagates

up to the service interface. If an error does not propagate up to the interface, it is named **latent error**. Once a failure occurs, it causes an internal fault of the computer system, or an external fault for another system being receiving a service from it.

Dependability Attributes

In [5], dependability is described as an integrating concept encompassing the following attributes.

Availability: defined by Recommendation E.800 of the International Telecommunications Union (ITU-T) as the "ability of an item to be in a state to perform a required function at a given instant of time or at any instant of time within a given time interval, assuming that the external resources, if required, are provided" [68].

Reliability: defined by the ITU-T Recommendation E.800 as the "ability of an item to perform a required function under given conditions for a given time interval" [68].

Safety: defined as the absence of catastrophic consequences of a failure on the users and on the environment.

Integrity: represents the absence of improper system alterations.

Maintainability: is the ability of a system to be maintained, usually if a failure occurs, by means of a repair strategy (e.g. repair or replacement of a broken component).

To assure, or improve, the attributes of the dependability, several **means** can be adopted.

Fault forecasting can be adopted to estimate the number, incidence and possible consequences of faults. The occurrence or the introduction of faults can be warded off by means of **fault prevention**. To reduce the number of and severity of faults, **fault removal** can be adopted. **Fault tolerance** allows the system to avoid or put up with the service failures that can be caused by the activation of a fault.

Dependability Indices

For quantifying dependability attributes, more formal definitions and metrics are to be introduced [48].

Let X be a random variable representing the time to failure (TTF) (or lifetime) of a system. Such a random variable can be characterized by the cumulative distribution function (CDF), the probability density function (pdf), or the hazard rate function h(t).

The **CDF** of the (non-negative) random variable X is simply defined as

$$F_X(t) = P(X \leq t), 0 < t < \infty \tag{2.2}$$

The **pdf** of X is defined by

$$f_X(t) = \frac{dF_x(t)}{\mathrm{d}t} \tag{2.3}$$

The **hazard rate** (or instantaneous failure rate) is defined by

$$h(t) = \frac{f(t)}{1 - F_x(t)} \tag{2.4}$$

Given the previous definition, for a time interval $(t_0, t_0 + t]$, the **reliability** $R(t|t_0)$ defines the probability that a system survives in this interval given that it was properly

working at time t_0. If $t_0 = 0$, $R(t|0)$ defines the probability that a system is up until time t,

$$R(t|0) = R(t) = P(X > t) = 1 - F_X(t) \tag{2.5}$$

The **conditional reliability** defines the probability that a system survives in the interval $(t_0, t_0 + t]$, given that the system has survived until time t_0:

$$R_{t_0}(t) = \frac{R(t_0 + t)}{R(t_0)} \tag{2.6}$$

The expected life of a system is defined by the **mean time to failure** (**MTTF**) as:

$$E[X] = \int_0^\infty t f(t) \mathrm{d}t \tag{2.7}$$

While the reliability represents the probability that a computer system is failure-free during a time interval, the **availability** represents the probability of a system being failure-free at a precise instant of time. Introducing the indicator random variable $I(t)$, which is equal to 1 when the system is up and 0 otherwise, we have the following definitions.

The **instantaneous availability** (or point availability) is the probability that a system is up at time t,

$$A(t) = P(I(t) = 1) \tag{2.8}$$

Note that the instantaneous availability $A(t)$ is always greater than or equal to the reliability $R(t)$ and, if the system has no repair or replacement strategies, $A(t) = R(t)$.

The **steady-state availability** (or limiting availability) is the value of $A(t)$ as t approaches infinity,

$$\lim_{t \to \infty} A(t) \tag{2.9}$$

Under very general conditions, the steady-state availability can be computed as:

$$A = \frac{MTTF}{MTTF + MTTR} \tag{2.10}$$

where the **mean time to repair** (**MTTR**) includes both the time to detect a failure and the time to repair it.

The **mean time between failures** (**MTBF**) is the sum of MTTF and MTTR:

$$MTBF = MTTF + MTTR \tag{2.11}$$

The **interval availability** (or average availability) is the expected fraction of time the system is up in a given interval of time $(0, t]$. It is defined as

$$A_I(t) = \frac{1}{t} \int_0^t A(x)\mathrm{d}x \tag{2.12}$$

It is easy to demonstrate that, when both limits exist:

$$A = \lim_{t \to \infty} A_I(t) = \lim_{t \to \infty} A(t) \tag{2.13}$$

2.2 Performance and Dependability: Performability

In 70s, researchers started to evaluate the effect of failures on the performance of the system. The common idea was of a failure causing the interruption of the computer system's service. However, if fault tolerance is adopted, even in presence of component failures, the system may be able to provide a subset of the services implementing the functions included in the

functional specification, or a subset of such services may be provided with a reduced quality. Examples of reduced quality can be slow service, limited service, emergency service. The system is said to be a *gracefully degrading system.*

Assuming that at the initial state the system is operating at its maximum performance, when a failure occurs, system performance may degrade. This kind of system offers several levels of performance. As an instance, a double processor system may be able to work, even if with a halved throughput, if only one of the processors fails. When the system is not performing at its maximum because of a partial failure of its functionality or performance, it is said to be in a *degraded mode.*

Beaudry was the first author proposing to develop measures which provide the trade-off between reliability and performance of degradable systems [8]. Then, Meyer introduced the concept of **performability** combining the performance and reliability/availability measure of a system [89].

2.2.1 Performability Indices

Let S denote the set of all possible configurations in which the system can perform its activity, and let $\{X(t), t \geq 0\}$ on S define a continuous time stochastic process describing the structure of the system at time t. Let $\pi_i(t)$ be the probability that the system is in state $i \in S$ at time t and π_i be the probability that the system is in state $i \in S$ as t approaches infinity. We can associate a reward rate to every state indicating the performance level offered by the system in that state. r_i represents the reward obtained per unit time spent

in state $i \in S$.

The **Reward Rate** at time t is indicated as $Z(t) = r_{(X(t))}$. It can be shown that $L_i(t) = \int_0^t \pi_i(\tau)\mathrm{d}\tau$ is the expected total amount of time spent by the system in the state i during the interval $(0, t]$.

The **Expected Instantaneous Reward Rate** at time t is given by

$$E[Z(t)] = \sum_{i \in S} r_i \pi_i(t) \tag{2.14}$$

The **Expected Steady State Reward Rate** of the system is

$$E[Z] = \lim_{t \to \infty} E[Z(t)] = \sum_{i \in S} r_i \pi_i \tag{2.15}$$

The **Accumulated Reward** in $(0, t]$ represents the total amount of work performed by the system during the interval $(0, t]$:

$$Y(t) = \int_0^t Z(t)\mathrm{d}t \tag{2.16}$$

Then, the **Expected Accumulated Reward** in $(0, t]$ is the amount of work done by the system during the interval of time $(0, t]$:

$$E[Y(t)] = E[\int_0^t Z(t)\mathrm{d}t] = \int_0^t E[Z(t)]\mathrm{d}t = \sum_{i \in S} r_i \int_0^t \pi_i(t)\mathrm{d}t = \sum_{i \in S} r_i L_i(t) \tag{2.17}$$

To better understand these metrics, consider the simple example in Figure 2.3. A system has three possible states. While it is in state 1, it has a reward rate equals to 2, that is, the performance level is "2". If a failure occurs, it goes to state 2, where its reward rate is 1. If

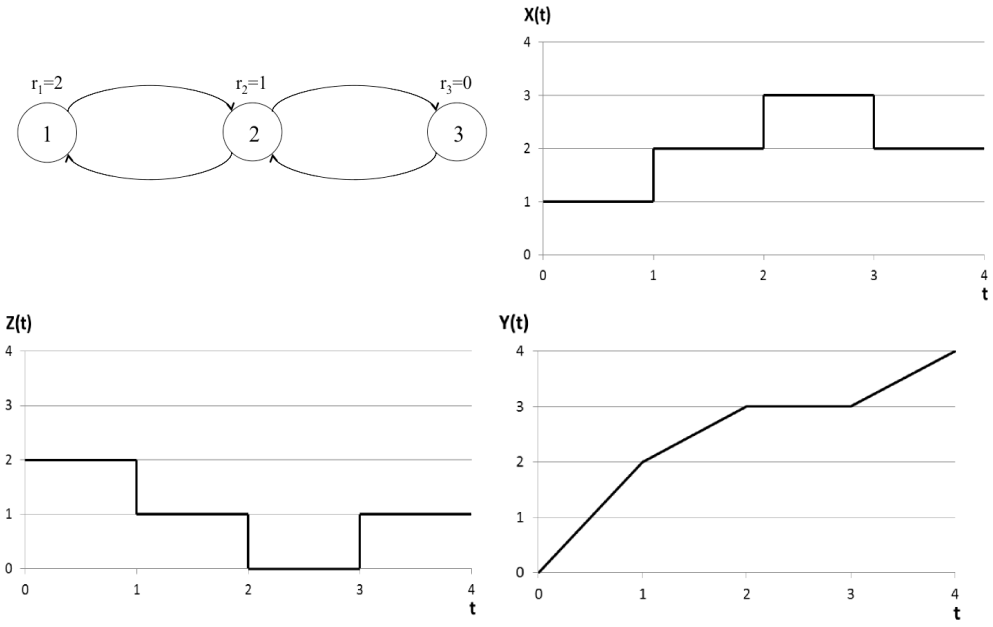


Figure 2.3: Markov reward model, evolution of the system over time (X), reward rate (Z), accumulated reward rate (Y).

the failure is repaired, the system goes back to state 1. If a failure occurs while the system is in state 2, it goes to state 3, where the reward rate is 0. In case of repair, the system goes back to state 2. This model is called Markov Reward Model (MRM). $X(t)$ represents the state of the system at time t and $Z(t)$ is the corresponding reward rate of the system. $Y(t)$ is the accumulated reward over the time interval $(0, t]$.

2.2.2 Performability Examples

The concept of performability has been often applied to the analysis of processing systems, above all when also fault tolerance mechanisms are adopted. From a pure availability/reliability analysis or a pure performance analysis the attention moved to their joint

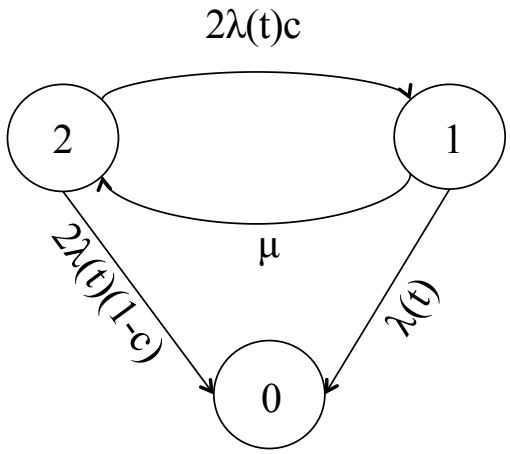


Figure 2.4: Two-processor-parallel-redundant system with imperfect coverage and with repair.

study. This is commonly done by means of **performability models**.

As a performability example, consider a two processors parallel redundant system with imperfect coverage [48]. The system has two processors with the same time-independent failure rate and single repair facility with a certain repair rate. Imperfect coverage means that not all individual processor failures are recovered from. If a covered failure occurs in one of the two processors, the other one continues working and the system is up, although in a degraded mode. If a not-covered failure occurs, the system fails.

Let $P1$ and $P2$ be the two processors; both of them have the same time-dependent failure rate $\lambda(t)$, independent on the state or the task of the processor. The coverage factor, denoted by c, represents the proportion of individual processor failures from which the system can automatically be recovered. Once $P1$ (or $P2$) fails, if the failure is covered by the recovery strategy, the system can properly work, with only one processor, with degraded performance, otherwise it fails. The processor repair facility is characterized by a constant

repair rate μ. The system can be modeled as in Figure 2.4: *State 2*: both processors are properly working; the failure rate of each processor is $\lambda(t)$, hence the equivalent failure rate of this state is $2 \cdot \lambda(t)$. If a failure covered by the recovery strategy occurs (rate $2 \cdot \lambda(t) \cdot c$), the system goes to *State 1*, otherwise to *State 0* (rate $2 \cdot \lambda(t) \cdot (1-c)$); *State 1*: one processor failed because of a covered failure; the system can continue working in degraded mode with one processor. In this case, two possible scenarios are possible: the failed processor is repaired (with constant repair rate μ) before the working one fails, then the system goes back to *State 2*; or the working processor fails (rate $\lambda(t)$), and thus the system fails (*State 0*); *State 0*: the system is down because either both processors failed, or a non-recoverable failure occurred. This is an absorbing state (the system can not be repaired if this state is reached).

If (i) the repair is minimal, i.e., the repaired processor is in a state (age) equal to the one before its failure and (ii) the repair time is negligible compared to time to failure, the model in Figure 2.4 is a non-homogenous continuous time Markov chain (NHCTMC).

For performability analysis, rewards are attached to the states of the Markov model by counting the number of active processors. In State 2, two processors are working, and the reward rate is equal to 2; in State 1, where only one processor is working, the reward rate is 1. A reward rate equals to 0 is associated to the absorbing State 0. Hence, the reward rates are $r_2 = 2$, $r_1 = 1$, $r_0 = 0$.

In Figure 2.5 is depicted the trend of the expected instantaneous reward rate of the model for $\mu = 120$, $c = 0.9$, and $\lambda(t)$ a Weibull failure rate with parameters $\alpha = 2.1$ and $\beta = 1.02$. Such a measure is very important in order to quantify the performance of the

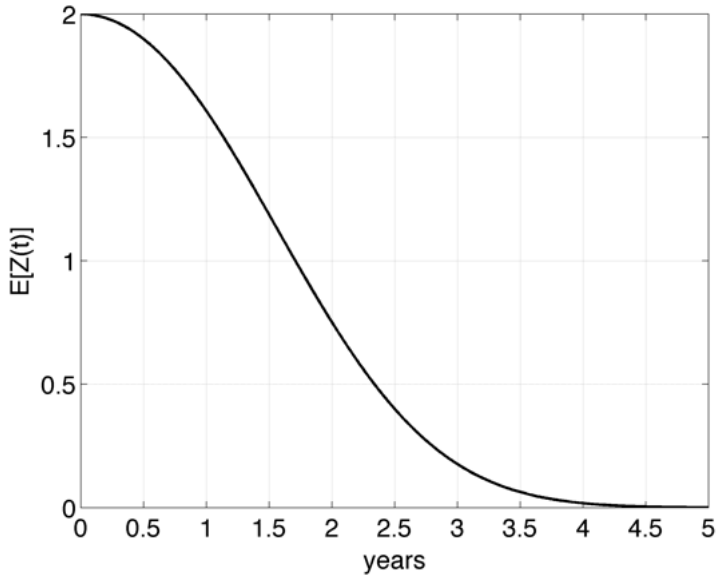


Figure 2.5: Expected instantaneous reward rate.

system in presence of failures. For instance, suppose the reward rate being the MIPS of the processors and that it is required for the system to perform at least 0.5 MIPS. The expected instantaneous reward rate shows that after about 2.3 years the system is no longer able to provide the desired service level. Hence, a renewal strategy is necessary.

In [135], analytic performability models are built to compare different rejuvenation policies. Since assuming both deterministic and exponential transitions, deterministic and stochastic Petri nets (DSPNs) are used, which correspond to an underlying Markov regenerative process (MRGP) that can capture both exponential and deterministic distributions. System performability measures are derived from the DSPN models as:

$$Perf = \sum_{c \in \Omega} r_c \cdot \pi_c \tag{2.18}$$

where π_c is the probability that the MRGP underlying the DSPN model is in a certain state

c, r_c is the system performance index under state c, and Ω is the state space of the MRGP.

2.3 Performance, Consumption and Dependability: Consumability

Since performance, consumption and dependability aspects appear peculiar for the evaluation of the efficiency of processing systems, we propose to simultaneously deal with them. The idea is to unify the analysis of the three aspects in a measure providing information about the produced quantity of work and the electricity required for such a production, considering also the failures that commonly happen. Hence, *the **consumability** is defined as the ability of a processing system to produce useful work in a certain time with energy efficiency, that is to properly exploit the incoming electric power given the inevitable occurrence of failures.*

The relation of dependability aspects with energy consumption or both consumption and performance is quite unexplored.

The connection between **consumption and dependability** is sometime studied as the cost of maintaining desired levels of quality of service (QoS) or as a different perspective of performance. As an instance, in place of evaluating the impact that an energy efficiency technique has on performance, its effect on the possible violation of service level agreements (SLAs) is assessed. This approach does not really evaluate the cost of a failure in terms of wasted energy, but the performance decrease due to some management techniques.

Cloud architectures are often accounted as a means for improving the efficiency of processing systems. Bashoon in [6] discusses the concept of a framework for the dynamic self-optimization of such architectures taking into account the trade-off between performance and consumption aiming at maintaining acceptable dependability requirements, which are evaluated with respect to the QoS. The framework is based on the Dynamic Data Driven Simulation Systems (DDDAS) paradigm. It uses data collected from the system to perform runtime simulations predicting system behavior and to identify the configuration to be used. This approach is similar to the one discussed in Section 1.2.3 and outlined in Figure 1.6. The idea is to merge a monitor of a system and a predictor, in order to tune some parameters for achieving the desired levels of consumption and quality.

Resource allocation for reducing both consumption and SLA violations is discussed in [49]. In this case, workload analysis is used to determine the typical demand for a service. A dynamic programming algorithm captures the common load of the system and helps in figuring out the best allocation for minimizing costs and risks from a provisioning perspective. The provisioning system is implemented, validated, and experimented through simulation. The results show that this approach can achieve up to 35% energy consumption reduction and reduce SLA violations by 21% with respect to other policies based on prediction.

In [15], authors present ASTRO, a tool based on stochastic models to support data center designers with information on sustainability impact, cost and dependability of cooling infrastructures. For the dependability, two main metrics are employed: Reliability Importance (RI), for identifying the most critical components, and Reliability and Cost Importance (RCI) relating the RI to the cost of components. Downtime is used for the

availability. Consumption is modeled considering the energy flow between system components and evaluated as the maximum energy that each component can provide (in the case of electrical devices) or the maximum cooling capacity (in the case of cooling devices). The system is modeled by means of Stochastic Petri Nets (SPN) and Reliability Block Diagrams (RBD). As a case study, authors show the use of the tool for the analysis of real-world data center cooling architectures. Although specific to cooling systems, this paper provides interesting hints on modeling for consumption analysis.

The relation between reliability and energy consumption is treated indirectly in [40]. In this case, the management of the temperature in data centers is evaluated by analyzing a large collection of field data from different production environments. The starting point of the paper is the evaluation of the optimal temperature at which a data center should work. Increasing data center temperature may reduce the energy consumption (due to the cooling system), but may also worse the system reliability or performance (due to device throttling). Data analysis confirms such intuitions. There is a correlation between the failure rate and the temperature. In the case of disks, the variability in temperature, more than the average value, has an effect on failure rates. As for DRAMs, no relation is found. Also the relation with **performance** is assessed. Specifically, data shows that the temperature may affect the throughput of I/O bound applications. As for CPU bound applications 3-4% of throughput decrease may be experienced. Results are computed through the analysis of an implemented system and provide significant hints for the modeling. Since based on pure measurement analysis (see Section 3.2), it is complementary to the model-based evaluation adopted in this dissertation.

The cost of fault tolerance techniques in terms of energy consumption is evaluated in [88]. The authors consider different configurations of checkpointing (checkpoint/restart, message-logging and parallel recovery) and their cost. An analytical model is used for evaluating their impact on a large data center, but the cost of the failures is not evaluated. Parallel recovery appears as the technique requiring less energy, up to reducing system consumption by 17%. Also, it comes out as the best technique from the performance point of view, allowing the system to complete the execution in a time reduced by 13%.

The simultaneous study of **performance, consumption and dependability** aspects, which we named *consumability analysis*, is often conducted for the hardware.

In [32], authors point out the importance for benchmarking techniques related to performance, dependability, and energy consumption of hardware systems. The focus is on the use of third-party components, for which one may need to quantify some properties for comparison and selection, such as performance, power, cost, and failure in time. The proposed approach aims at identifying suitable measures to characterize and compare the components and defining experimental procedures for measurements. In [39], authors state that reliability, low energy consumption, and high performance are the three design objectives for on-chip networks. The paper analyzes the impact of various error-control schemes on the trade-off among reliability, performance and energy when voltage swing varies. As a performance metric, it is used the probability to transmit a certain quantity of useful bits during a time interval in the presence of noise. Energy consumption is also taken into account and by means of simulations the authors estimate the consumption of the error

control circuits when failures occur. Dependability, power, and performance trade-offs for multicore architectures are debated in [117]. The proposal is to adapt processor resources according to the requested performance and soft error rate. A performability model also including consumption measures for wireless sensor networks is presented in [36]. The paper presents a framework for the assessment of WSNs based on the automated generation of analytical models. Those models can be used to drive design choices. In this case, the consumption represents an aspect of the sensor behavior which may cause failures nevertheless. How failures affect the consumption is not contemplated.

2.4 Discussion

Over the years, scientific literature has proposed many works about the three aspects of performance, consumption and dependability, separately. In other cases, performance is assessed together with availability/reliability or consumption, introducing the aspects of performability and energy efficiency, respectively. Nevertheless, the relation between dependability and consumption is not well explored and even less its relation with both energy consumption and performance. In a scenario where an administrator has to improve the performability of a system, s/he does not know the impact a proposed technique and/or technology may have on the energy consumption of a system. Similarly, if the energy efficiency is to be improved, the possible effect on a dependability metric is neglected.

Based on this observations (which are supported by figures reported in Chapters 5 and 6), ***this thesis introduces the concept of consumability***. Its measurement is

paramount for administrators to consciously select designs and tunings of the processing systems to achieve the desired trade-off among costs (in terms of electricity consumption), performance (as the quantity of useful work produced for final users within a certain time), and availability (meant as the readiness to produce the useful work).

By knowing the power consumption and performance degradation caused by failures, it is possible to decide whether it is worth putting (expensive) fault tolerant mechanisms on the field. Also, approaches for resource management can be properly tuned for reducing energy consumption while also avoiding performance degradation and, *at the same time*, availability issues.

The evaluation of the three aspects separately, of performability and of energy efficiency has been carried out in several ways. Some of the discussed papers are based on measurements, that is, the value of useful parameters are collected directly from the system and properly analyzed. While valuable for the assessment of existing systems, these approaches are limited for evaluating the impact of other configurations, since they would require their actual implementation. Hence, model based approaches are commonly adopted to overcome such issues. Measurement- and model-based approaches are discussed in the following Chapter 3.

Chapter 3

Evaluation Approaches

In previous chapters, the main aspects of processing systems have been introduced. Also, from the examined literature, it is clear the importance of evaluating such aspects with reference to some metrics of interest. This evaluation can be performed using several approaches, generally classified as measurements-based or model-based. Model-based approaches avoid unnecessary details and system implementation, allowing one to measure some parameters with an acceptable approximation within affordable cost and time. However, not all modeling approaches are scalable. In the case of large processing systems, hierarchical modeling and simulation are more advisable.

3.1 Evaluating Processing Systems

Processing systems are usually evaluated with respect to performance and availability/reliability aspects. Nowadays, energy consumption is considered as well.

The process of evaluation consists in the definition of *what* one want to measure and in assigning it a value (the measuring itself). Formally, a *metric* is a function defining the distance between the elements of a set. The measurement is commonly performed with the direct observation of a system. For performance, one may count the number of operations the system performs in a certain time interval. As for the consumption, power distribution units (PDU) can be used to achieve the instantaneous power absorption of the various system components. If one want to quantify how much a system is reliable, failure related data

can be collected and analyzed in order to derive a value for one or more metrics (e.g., mean time to failure, distribution of the failures over time). Approaches of this kind are generally classified as **measurements-based**. They are largely adopted and trusted as based on real operational data. However, direct measurement is also costly and time consuming since requiring the system implementation, data collection, filtering and then analysis. On the opposite, **model-based approaches** offer an abstraction of the real system that does not include unnecessary details and, above all, system implementation can be avoided. If the aim is to assess a particular design of a processing system, the metrics can be estimated through a model mimicking its expected behavior.

3.2 Measurement Based Approaches

Measurement of processing systems includes monitoring and recording useful data, which can be classified in three broad categories: performance, environment, and operational data. **Performance data** are among the most monitored, and are related to the use of system resources. Main examples are the usage of CPU or memory. Other sources are about the use of the network, such as inbound and outbound traffic. **Environment data** are rarely exploited, even if they could be very useful to justify some system trends. They describe the status of the environment in which the system is placed and node parameters not related to performed activities. In this category fall temperature, humidity and the status of cooling systems. The monitoring of the energy consumption is also in this category. Finally, **operational data** encompass all the information achieved by collecting, and presumably

parsing and filtering, logs from the various software layers of the system, including event logs produced by applications and by the OS. A typical log entry contains a time stamp and a text field with information of interest (e.g., an operation performed by the system, the scheduling of a job, the occurrence of a failure).

The collected data can then be analyzed both in a prototype stage and in the operational stage [71]. In the **prototype stage**, the behavior of the system under particular circumstances is evaluated. As an instance, the energy consumption under different loads can be assessed for consumption analysis. For reliability analysis, fault injection can be used. It can provide information about the system behavior when faults are activated, from fault occurrence to failure detection and to system recovery. Stress testing also can be used to study the system response beyond normal cases. However, these approaches can not be used to evaluate the metrics related to the occurrence of failures, such as the mean time between failures (MTBF).

To evaluate the actual behavior of the system under real load and circumstances, measurements are done during the **operational stage**. These data contain information about occurring failures and consumption related to the load of a specific system and conditions in which it naturally operate.

The analysis of measured data is commonly based on statistical techniques.

Workload analysis is the basis for evaluating the performance of a processing system [16]. In the case of batch systems, the jobs are sequentially processed without any interaction system-user. In this case the load is commonly described from a static viewpoint

by applying classification techniques (e.g., clustering) grouping jobs with homogeneous resource consumption. Then, each group of jobs is characterized, for instance, with respect to inter-arrival times (time between the submission of two consecutive jobs) or completion times (time required by the system to execute the job). In the case of interactive systems, the load is described including its dynamics to take into account time-varying characteristics. By way of example, also the user behavior can be considered to describe the expected load of the system [134].

Consumption analysis is commonly performed on data collected at various levels of consumption [23]. In the case of a server, the idle power is the power consumed by the server when none of the applications are running on it; it is also referred to as static power. The dynamic power is the one consumed when applications are being executed. This can be divided in CPU power, due to applications that mainly use CPU, and I/O power, due to applications that mainly use I/O devices (e.g., disks and network).

Field Failure Data Analysis (FFDA) is an effective mean to gain knowledge about failures occurring in a system [34]. The most adopted field data sources for FFDA are event logs and human-generated failure reports, largely adopted in the literature for characterizing failure and repair distributions of processing systems during their operation [118, 99, 35]. Log-based failure analysis is usually performed through (i) data filtering, concerning the removal of log events not related to failures, (ii) data coalescence, used for grouping redundant or equivalent failure events, and (iii) data analysis, regarding the evaluation of dependability measurements. These steps may introduce errors as in the case of *collisions* (events related to two different failures are grouped as related to a unique failure) and

truncations (events related to the same failure are accounted as related to more failures). Failure reports are compiled with information related only to failures by technicians that have direct access to the system. Such reports do not require any preprocessing activities to carry out the analysis. Nevertheless, human interpretation and possibly errors can be included.

The application of such analysis to real cases is shown in Chapters 5 and 6.

3.2.1 Examples of Measurement Based Evaluation

Analysis of data about reliability and energy consumption is treated in [40], where the relation among failures, temperature and performance in data centers is evaluated by analyzing a large collection of field data from different production environments. This is an example of measurement based analysis to support the tuning of the cooling system creating the potential for saving energy while limiting reliability issues. Obviously, it is useful only for tuning devices already in use and for each tuning the analysis should be repeated to assess its effectiveness. For other devices or management techniques, as common data analysis approaches, the utility is restricted. To estimate the cost of performance decrease due to failures, one should estimate the cost of maintaing a certain temperature in the system. Even assuming this estimate being reliable, it is neglected the consumption due to failures not related to the temperature increase.

Evaluation based on fault injection is presented in [32] for the benchmarking of third-party components. Also in this case performance, consumption and reliability are jointly

assessed. After a fault free execution (*golden run*), faults are injected, and the various metrics are evaluated. The approach is interesting, but (i) it is not useful to estimate the failure rate of the system, since failures are driven by the fault injection, (ii) application on a running data center providing a service is not so easy, since unlikely one can inject faults in a running system, and (iii) if we want to evaluate different techniques (of fault tolerance or management), it is unfeasible to implement and test all of them. The approach can be certainly adopted for studying failure costs on a single server and then estimate the impact on the whole data center. Cost of failures due to the interaction among the servers is then to be studied separately.

Benchmarking tools are commonly adopted for acquiring useful information characterizing a system. Some of them are discussed in Section 1.2.3. SPEC corporation also provides the guidelines for the benchmarking of a computer system [122]. They propose SPEC_power, a benchmark to measure both performance and power measures. This is accomplished by driving the workload in a controlled manner, while other benchmarks can only be run at maximum performance. The motivation is that when under the maximum capacity utilization, the load is more variable and the benchmark should be able to capture such variations. SPEC_power can be executed at different performance levels so that the energy efficiency in different circumstances can be evaluated. Several load levels are defined as a percentage of peak workload. The evaluation procedure mainly consists in the following steps:

1. System is made ready to execute the benchmark and for measurement (e.g., nodes are made usable, sensors for energy monitoring are activated).

2. A performance level is selected.

3. The benchmark program is executed while power and thermal measurements are collected.

4. The benchmark completes and the collection of performance, power and thermal data ends.

5. The performance level is increased and the benchmarking restarts (from point 3).

6. When selected performance levels are covered, the data post-processing starts.

Two main concerns arise on measurement-based approaches. The first one is that they can be used only to evaluate what the system already has. If some management techniques (e.g., a scheduling strategy) or fault tolerance techniques (e.g., job replication) are to be evaluated, they should be implemented and then the benchmark executed. This is often unfeasible because of both the utilization of the system (which can not be stopped for such experimentations) and cost issues (someone should be payed for implementing a strategy that may be never adopted). The second concern is about benchmarking-based approaches. "System is made ready for measurement" [122] implies that the system is forced to be in a state that can be different from the ones it is in during the operational phase. By varying the performance, not all the possible states are covered and, above all, failure states are neglected. As a consequence, measured energy efficiency is only related to ideal situations and to the desired behavior of the system, while the failures that commonly happen are

completely ignored. On the contrary, by forcing some failures, only their cost can be accurately estimated but no information about the common operational behavior of the system is achieved.

3.3 Model Based Approaches

Model based approaches are preferred when parameters of interest can not be measured directly. It is not always possible to stop the system and equip it for measurement purposes, or to inject faults in a running system. Also, it is expensive to implement each configuration, tuning, technique one want to evaluate. As an instance, we already cited the article by Recupero appeared in Science [106], where the importance of analytical models for designing green network devices was discussed.

Modeling techniques are usually divided into two broad categories: *non-state-space* models [112] and *state-space* models [128]. The former model a system on the basis of its logical structure and how the availability of each component may affect the one of the system, assuming that components are independent of each other. Examples are reliability block diagrams (RBDs) and fault trees (FTs). The latter are often preferred because of their capacity of modeling more complicated interactions between components and different failure/repair behaviors. Among state-space models, Markov chains are widely used in availability/reliability studies. A variant called Markov Reward Models (MRM) was introduced for performability analysis [128]. We presented an example of MRM in Section 2.2.2. In the case of complex systems, these models have a large number of states and their creation

becomes error-prone. Stochastic Petri Nets (SPNs) can be used for the specification and automatic generation of the underlying Markov model. Stochastic Reward Nets (SRNs) are introduced in Section 4.2. Stochastic Activity Networks (SAN) were also introduced in [114] for facilitating performance and dependability evaluation.

3.3.1 Examples of Model Based Evaluation

In [54] a MRM for performance-consumption analysis of Cloud systems is presented. The model is created with an interacting sub-models approach for reducing complexity and chance of error of a single monolithic model. For modeling the power consumption of a virtual machine, its average resource utilization is assumed to be v_a and to each state of the CTMC it is assigned a reward rate $r = h_i + kv_a$, where k is the number of active virtual machines in a physical machine. Steady state power consumption in each physical machine is computed as expected steady state reward rate.

In [142], a computational cluster, which could be a part of a computational grid, is modeled as an open queueing network. For the workload, following assumptions are done: (i) a job can be executed on any processor; (ii) jobs are non-preemptable (their execution can not be suspended until completion); (iii) job service times are unknown to the scheduler; (iv) jobs are CPU intensive. The job arrival rate (computed from the inter-arrival time between consecutive jobs) is assumed with exponential distribution. Several metrics are adopted as output of the model. *Response time* of a job is the time from the arrival to the scheduler to the time of job completion. The *slowdown* measures the time the job is in the

system but not running (e.g., time spent for scheduling or migration). *Energy consumption* is evaluated from the power and the time for which it is used.

In [88], an analytical model is used for evaluating the impact of fault tolerance techniques in terms of energy consumption on a large data center. Equations used for modeling both system performance and consumption are simplistic and can hardly describe the behavior of a real system. As a matter of fact, fault tolerance is assessed, but failures are not modeled.

The paper [39] analyzes the impact of various error-control schemes on the simultaneous trade-off between reliability, performance and energy when voltage swing varies in on-chip networks. As a performability metric, $P(L, T)$ is defined as the probability to transmit L useful bits during the time interval T in the presence of noise, since in an on-chip interconnect the useful work is to transmit useful bits. For the consumption, both static and energy consumption are studied. They are mainly function of transmitted/received bits and are analytically modeled. The proposed approach is interesting and may be applied to large processing systems. The consumption model is used both to evaluate the cost of the device and of each type of failure. Nevertheless, the whole system is modeled with an analytic model for the consumption, through which also the variations corresponding to different workloads are estimated. Also, only failures that can be treated by the error control circuits are considered, while imperfect coverage is not contemplated. In other terms, more than the cost of the failures, it is only the cost of a fault tolerance mechanism under certain circumstances.

In [36], a performability model for wireless sensor networks is presented. The model, based on SANs, also includes a PowerSupply model that mimics the battery consumption

of a node and the effects of the possible failures (e.g., node failure due to low power). The behavior of the components and the failures are modeled together. That is, the model of a node takes into account the sub-components (i.e., sensor board, power supply unit, communication layer, and routing) and the possible failures that they can experience. Failure modes are identified by means of a Failure Modes and Effects Analysis (FMEA). The adopted modeling approach is very interesting. The system behavior and consumption are first considered separately and then a model encompassing all the aspects is created. For populating the models, data-sheets and behavioral simulation are used.

3.4 Scalable Models for Large Processing Systems

Monolithic or one-level Markov chains are a typical modeling formalism, representative of the state-of-the-art in processing systems modeling. However, as the system size grows, the dimension of the state space increases. The growth of the state space as the model takes into account more details of the system is known as the **largeness problem** of Markov models [128]. Also when approaches for automating the generation of the underlying Markov model are adopted, the solution of large models is an issue. The use of a single monolithic model and analytic-numeric solution is not scalable for large Clouds due to the large state-space of the underlying Markov chain. To resolve this issue, a scalable stochastic modeling approach based on *interacting sub-models* can be used.

Many models are hierarchical in nature, that is the system can be modeled through various models, each for an aspect, and then composed in a unique model. The complexity

Table 3.1: Summary of comparison among the three modeling approaches: (i) single monolithic model, (ii) interacting sub-models, and (iii) simulation.

Comparison index	**Metric**	**Dimension**	**Monolithic model**	**Interacting sub-models**	**Simulation**
Accuracy	Downtime (sec)	21 PMs	3664.527	3664.527	3416.4
		300 PMs	-	49993	52402
Scalability	Max #PMs	-	≤ 21	> 300	> 600
	#states	21 PMs	$4,816,252$	$3,770$	-
	#states	300 PMs	-	$9,196,353$	-
Efficiency	Solution time (sec)	21 PMs	301.700	0.584	10.434
		300 PMs	-	364.025	294.346

and characteristics of large processing systems may lead to cyclic dependency among the sub-models, needing fixed-point iteration. Another solution to the largeness problem is shown where underlying large Markov model generation is avoided by directly solving the stochastic net by means of *simulation*. We demonstrated the effectiveness of interacting sub-models and simulation for solving very large state-space models in [53]. Table 3.1 summarizes the comparison among the three approaches: (i) commonly used single monolithic model, (ii) interacting sub-models, and (iii) simulation, in terms of accuracy, scalability, and efficiency. Results are related to the SRN availability model of a cloud architecture, selected as a representative of modern large data centers, where a large number of physical machines (PMs) are used. We used the SPNP software package [62] to solve the SRN models on a desktop PC with 3.0 GHz CPU and 4 GB memory.

The numeric solution of the monolithic model can be computed for a data center with up to 21 PMs. For larger values, the desktop PC presented memory overflow issues. The interacting sub-models approach is rapidly solved numerically up to 300 PMs and its solution is accurate with reference to the monolithic model. The error on the mean values introduced

by the simulation is 4.8% with respect to the numeric solution of the interacting sub-models, when considering a data center with 300 PMs. However, results achieved by means of analytic-numeric solution of the interacting sub-models are in the corresponding confidence intervals achieved when simulating the monolithic model. When the system has more than 300 PMs, simulation also appears to be more scalable than the numeric solution of the interacting sub-models. In fact, the number of states of the underlying Markov chain in the interacting sub-models easily reaches 9 million, making this approach infeasible for very large systems. By contrast, simulation does not require to build the underlying state-space model and allows us to solve very large models with 19.2% reduction in solution time.

Chapter 4

Consumability Analysis of Batch Processing Systems

The consumability analysis is introduced for simultaneously evaluating performance, consumption and dependability aspects of processing systems. We adopt a model based approach for the analysis of batch systems. A consumability model is created by means of the composition of a performance model, mimicking the correct behavior of the system, a consumption model, considering the power static and dynamic components, and a failure model, encompassing the possible deviations from the correct service delivery. The model can be adopted for any batch processing systems supporting the administration in carrying out the consumability analysis. A metric is introduced to quantify the consumability.

4.1 Analysis Rationale

For jointly analyzing performance, consumption and dependability aspects of a processing system, we adopt a model based approach. A model allows us to study the behavior of the system under different circumstances; as an instance, we can assess the effect of different fault tolerance or management strategies while avoiding their actual implementation. However, the creation of a model that is able to mimic the actual behavior of a system requires an accurate study of the system itself. In this dissertation, we focus on batch processing systems. That is, systems where jobs run without end user interaction. We first look at

the system evolution by considering its behavior from the job execution perspective. As a matter of fact, in a system running batch jobs, both performance degradation and failures directly reflect on the jobs submitted to the system by its users, and a job, in turn, affects the consumption of the system. The problem is to identify the relations among performance levels, energy consumption dynamics, and failures. At this aim, we use a hierarchical approach; that is, performance, consumption, and failures are first modeled separately and then composed in an overall model which takes into account their mutual relationships (see Section 3.4). This requires to model the system evolution looking at how each possible state can affect performance and consumption and how failures can affect the evolution, in turn.

Given a system S, consider the set of all possible configurations in which it can work (properly, not properly and with different consumption levels) as a sample space Ω. In the case of a batch processing system, we assume configurations being dependent on the status[1] and number of jobs present in the system. A job $j \in J$ evolves through some statuses $\Sigma = \{\sigma_1, \sigma_2, \cdots, \sigma_m\}$, also including failure statuses. Each job j can be classified as belonging to a certain job type $K = \{k_1, k_2, \cdots, k_n\}$. We assume that the type of a job is *a-priori* known and that its evolution through statuses in Σ can be described by a stochastic process. Hence, the system evolves through the possible $\Sigma \times K \times J$ states, which form the state space Q of S. We assume that Q is discrete. The evolution of S can also be described by a stochastic process $X_S(t) = \{X(t) \mid t \in T\}$, where X is the random variable $X : \Omega \to Q$ and $T = [0, +\infty]$ is a set of observation times and may be either discrete or continuous.

[1] We use the term *state* for the system and for the checkpoint of the jobs, and *status* for the various stages of the job life cycle.

To each state both a performance level and a consumption correspond, depending on the number and type of jobs in each status.

From this formalization, it originates the modeling described in Section 4.2.

4.1.1 Steps of the analysis

The analysis is performed through the following steps.

- **Modeling of the system** (Section 4.2). We separately model performance, consumption, and failures of the system achieving three models that are then composed in an overall ***consumability model*** for the joint analysis of the three aspects.
  - **Performance model**. Given the correct execution of a job and its life cycle, we create this model accordingly.
  - **Consumption model**. The consumption of the system is modeled taking into account the static consumption of the system and the dynamic one, which depends upon the different kinds of running workload.
  - **Failure model**. In this model we include the deviations from the correct job life cycle that commonly happen in batch systems.

  Fault tolerance is also modeled to assess its impact on the consumability of a system.
- **Model populating** (Sections 5.2 and 6.2). For performing the analysis of a specific batch system, some inputs are to be provided to the model. These are the characterizing parameters of the system (e.g., job inter-arrival distributions, failure rates),

which can be estimated by means of data analysis techniques for workload, consumption and failures. For our case studies, we use data from a real batch system at the Federico II University of Naples; additional experiments and studies are performed for performance and failures in virtualized environments.

- **Model validation** (Section 5.3). The composite model is validated against data from a real batch system.
- **Simulation of the model and analysis of the results** (Sections 5.4 and 6.3). Finally, the model is solved by means of simulation to study the trend of performance, consumption and availability, and their relation, i.e. the consumability of the system.

4.2 System Modeling

For the modeling, we use a variant of Stochastic Petri Nets (SPNs) called Stochastic Reward Nets (SRNs), which automates the construction and solution of the underlying Markov model [26].

Stochastic Reward Nets (SRNs) mainly consist in places (represented as circles) and transitions (represented as bars). *Places* represent the various conditions that hold in the system. *Transitions* represent the various events that could occur in the system. *Arcs* (represented as directional arrows) are drawn from places to transitions and from transitions to places. An arc drawn from a place to a transition is called an *input arc* into the transition

from the place. An arc drawn from a transition to a place is called an *output arc* from the transition to the place. In each place, there can be zero, one or more *tokens* (drawn as filled dots). The presence of a token indicates that the corresponding condition holds.

The distribution of tokens in the various places of the SRN is called the *marking* of the SRN. A marking constitutes a *state* of the net.

A transition is enabled in a marking if each of its input places contains at least one token (or more, if the corresponding arc has a *cardinality* different from 1). An *inhibitor arc* from a place to a transition implies that the transition is not enabled if the corresponding inhibitor place contains a token (or more, depending on the arc cardinality). Also a *priority* can be associated to a transition. It is an integer specifying that a transition may be enabled only if no higher priority transition is enabled. Each transition may have an associated (boolean) guard function (or enabling function) g defined on the set of possible markings, and the transition is enabled in the marking M only if the function returns true in that marking, $g(M) = true$. The guard function g of transition t is evaluated in mark M when (i) no transition with priority higher than t is enabled in M, (ii) the number of tokens in each of its input places is larger than or equal to the cardinality of the corresponding input arc, and (iii) the number of tokens in each of its inhibitor places is less than the cardinality of the corresponding inhibitor arc.

If enabled, a transition can *fire*; this corresponds to the happening of the event associated with it. The firing of a transition causes the flow of a token (or more, depending on the cardinality of the arcs) from each of its input places to each of its output places, resulting in a new marking. The events associated with a transition take a period of time to happen.

In SRNs this is described by associating a *firing time* to each transition. The firing time is modeled as a random variable characterizing the time that elapses from the time point at which the transition becomes enabled to the time point at which the transition actually fires. An *immediate transition* (represented as a thin bar) can be viewed as a timed transition whose firing rate approaches infinity: when enabled, it requires zero time to fire.

A marking is reachable from another marking if there is a sequence of transition firings which results in the new marking. The set of all the markings together with the transitions among them is the *reachability graph* of the SRN. It is isomorphic to a continuous time Markov chain (CTMC). In presence of immediate transitions, the reachability graph presents vanishing markings, where the SRN does not spend any time, since they enable immediate transitions which fire as soon as they become enabled.

The metric of interest is computed by assigning appropriate *reward rates* to the states of the SRN.

4.2.1 Performance Model

The performance model describes the behavior of the batch system without considering failures, but the correct execution flow of a batch job, as depicted in Figure 4.1a. This is the execution flow according to Torque, a popular resource manager for batch systems [1]. A job submitted to the system is *queued*; then it becomes *running* when resources for its execution are identified and its execution time arrives; when the execution terminates, the job is *completed* and results are returned to the user during the *exiting* stage.

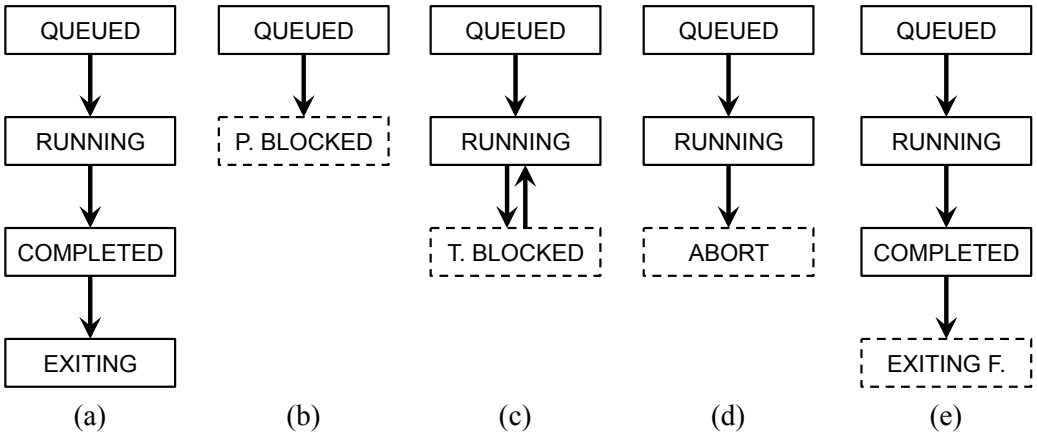


Figure 4.1: Status transitions of a job in a batch processing system. (a) correct execution; (b-e) execution with failures.

We model the life-cycle of a job with an SRN as depicted in Figure 4.2. Transition *arr* represents the job arrival. After the arrival, a job is queued and a token is added to place Q. Scheduling of a job is modeled by transition *sch*. Upon its firing, the job becomes running and a token is taken from Q and it is placed in R. Transition *cpl* depicts the completion of the job. At this point a token is removed from R and one is added to place C. Upon firing of transition *exg*, a token is removed from C, describing the exiting status of the job, i.e. the notification of the final result to the user that submitted the job. The guard function g_s is used for inhibiting the scheduling capacity when no enough free resources are available for executing a job. This depends on the number of already running jobs and on the total available resources. It is specialized for the particular case study.

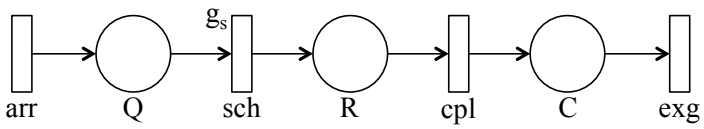


Figure 4.2: SRN representing the performance model of a batch processing system.

The probability distributions of inter-arrival, scheduling, completion and exiting times are associated to each transition in the model. The scheduling time distribution also includes the time that the job spends in the queue, hence it corresponds to the time interval from the job submission until the job execution starts. In the case different types of jobs $K = \{k_1, k_2, \ldots, k_n\}$ are identified in the system, the SRN is repeated n times as are the types of jobs. The guard function g_s has to take into account all the jobs running in the system, that is the number of tokens in the places R_{k_i}, each for each job type $k_i \in K$.

Inputs. This performance model takes as input parameters: (i) the number of available worker nodes, (ii) the maximum number of job of each type a node can execute, (iii) the inter-arrival time distribution, (iv) the scheduling time distribution, (v) the completion time distribution, and (vi) the exiting time distribution.

Outputs. The model returns as output the performance of the system. Different performance metrics can be used. The ones considered in this dissertation are the *throughput* of the system, as number of jobs completed in the time unit, or the *mean response time*, as the mean time to complete a job, since common metrics for evaluating batch processing systems (see Section 2.1.1). In the case of more types of jobs, the performance of the system is computed by summing the rewards for each net (e.g., the expected rates of the exiting transitions exg_{k_i} for each $k_i \in K$).

4.2.2 Consumption Model

For the consumption, we adopt an analytical model similar to the ones proposed in [124] and [10]. However, we see the data center as a whole. Thus, we do not create a model depending on the consumption of each server component, but we evaluate the consumption due to the types of workload present in the system.

Formally, the total consumption W_{tot} of the system is:

$$W_{tot} = W_s \cdot N + (\sum_{k_i \in K} W_{k_i} \times AR_{k_i}) \tag{4.1}$$

where $W_s \cdot N$ represents the static consumption of the system and $W_{k_i} \times AR_{k_i}$ is the dynamic power consumption due to running jobs of type k_i. The static consumption depends on the number N of worker nodes in the data center; K is the set of the types of jobs that are executed by the system, each of which has a consumption weight represented by W_{k_i} and an activity ratio AR_{k_i}, indicating the number of running jobs of type k_i. AR_{k_i} is given by the number of tokens in place R_{k_i}.

Inputs. This model takes as input parameters: (i) the number of worker nodes in the system N, (ii) the static consumption of each worker node W_s (in Watts), (iii) the consumption weights of each type of job W_{k_i} (in Watts), and (iv) the number of running jobs of each type AR_{k_i}.

Outputs. The model computes the consumption of the system (in Watts).

4.2.3 Failure Model

The performance model does not take into account the possible failures that may occur. It can be used for a pure performance analysis or, if composed with the consumption model, for a pure energy efficiency analysis as in [46, 54, 124, 15, 10], not reflecting the actual system behavior. To contemplate also dependability aspects and study the waste of energy due to failures, we model the possible failures that can happen in a batch system.

Failure characterization

Unfortunately, jobs not always evolve through the statuses depicted in Figure 4.1a, due to the occurrence of failures. From the perspective of the job execution, some failures can hinder the correct execution and completion. Hence, given the performance model discussed in Section 4.2.1, we consider deviations from that life cycle. Of such deviations we also found correspondence to actual failures that happen in batch systems from the literature, from trustworthy web resources and from the analysis of the system employed as a case study (see Chapters 5 and 6). We do not take into account those failures due to user mistakes. There is no way for a system maintainer to reduce the number of such failures nor the related consumption, indeed. Furthermore, the deviation from the correct service and possible SLA violations are not ascribable to the service provider.

Queue and running failures. One of the known issues affecting batch systems is represented by permanently or temporarily blocked jobs [108, 67]. In the case of *permanently*

blocked job, after being queued, the job enters the blocked status and its status does not change anymore (Figure 4.1b). Since this happens while a job is queued, affecting the scheduling transition and inhibiting the job becoming running, we name the related failure **queue failure**. A *temporarily blocked job* consists in a job whose status changes from running to blocked and to running again after a certain time. Such a situation is depicted in Figure 4.1c. Since affecting a running job, we name the related failure **running failure**.

We found degradation trends ascribable to such failures by means of performance analysis on a real batch processing system, the S.Co.P.E. scientific data center (which is introduced in Section 5.1).

We analyzed some performance and workload data by applying a specific combination of statistical hypothesis testing techniques as illustrated in [30]. To quantify the workload of the system, we used the *number of jobs submitted to the system at each day*. To quantify performance, we computed the *average duration of jobs at each day*, that is, the mean value of the duration of the jobs completed at each day.

We found some performance degradation trends on a small subset of nodes and queues in the system. The average slope of the trends was 0.3358 hours/day, that is, the average job duration increased by a fraction of hour at each day. In many cases only a rigorous statistical approach is able to detect the increasing trend of the average daily job duration. An interesting finding was that the detected trends at different nodes occurred at overlapped periods of time. Moreover, the performance data seem to have followed similar patterns.

The similar trends found on several nodes are a signal that the software at those nodes were suffering from a common cause of performance degradation. We also performed an

analysis on the types of jobs. This analysis did not find a relation between the type of job and the presence of a trend, since performance degradation trends manifest both in CPU-bound and I/O-bound jobs. This suggested that the basic software infrastructure, rather than user software, is the likely root cause of the performance degradation.

From a more detailed analysis of the job life cycle and a discussion with the system technicians, we heard that the most frequent reason for those behaviors was the accumulation of errors in a distributed file system, whose access privileges are modified to read-only, therefore some worker nodes are not able to execute jobs, even if they appear to be up and properly running. In fact, read/write errors, due to filesystem corruption, caused the status transition between *running* and *blocked* of several jobs, thus delaying their completion. The accumulation of blocked and restarted jobs, also caused the shortage of computing resources, such as memory, which delayed the execution of other jobs. These errors were more and more frequent with time, thus causing a gradual performance degradation of the nodes accessing that file system and hampering the proper execution of jobs. The problem lasts until a maintenance intervention (the reboot of the nodes), which interrupts the performance degradation trend.

Aborts. The execution of a job can be suddenly interrupted because of the occurrence of some failures in the machine where the job is running. This situation is depicted in Figure 4.1d. We refer to such an event as an **abort** of the job. Possible causes for these failures are the same that prevent the execution of queued jobs (causing the job transition to the blocked status). Other causes are the bailout or reboot of nodes.

As a matter of fact, we applied log analysis techniques, introduced in Section 3.2, to the system and scheduler logs of the real batch system S.Co.P.E. We found some entries related to killed jobs ("LOG_ERROR::node_bailout [...] job will be killed").

Exiting failures. In the last stage of the job life cycle, after the completion of a job, results may not be returned to the user because of an **exiting failure**. This is depicted in Figure 4.1e.

Also in this case, we found occurrences in the S.Co.P.E. data center. Among the filtered logs, some entries were related to the failure in returning the results to final users ("LOG_ERROR::sys_copy [...] Output failed"). They are due to misconfigurations on the worker nodes preventing the secure ssh connection with the machine to which results must be returned.

More detailed analyses of these failures in the S.Co.P.E. system are discussed in Sections 5.2.3 and 6.2.3.

Discussed failures are modeled as in Figure 4.3 and detailed in the following.

Figure 4.3a models the **queue failure**. The firing of transition qf removes m tokens from place Q and adds a token to QF. The repair is modeled by transition qr that, upon firing, removes a token from place QF and adds m tokens in place R. The multiplicity m models the number of jobs that, at the same time, permanently become blocked. This is due to the propagation of the malfunction and to the number of jobs running on a worker node

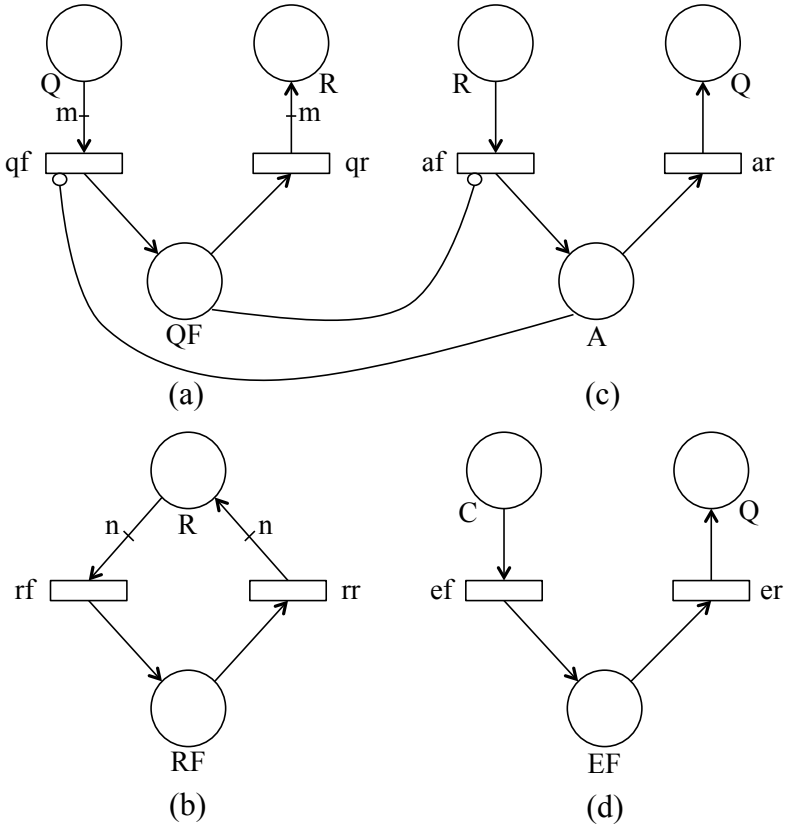


Figure 4.3: SRNs modeling (a) queue failures, (b) running failures, (c) aborts, (d) exiting failures.

(if the failure is due to a machine on which m jobs have been scheduled, m jobs experience the failure).

Figure 4.3b shows an SRN modeling the **running failure**. rf represents the transition of a job in the temporarily blocked status: upon its firing, n tokens are removed from place R and one token is added to place RF. Transition rr models the job becoming running again, moving a token from RF and adding n tokens back to R. The multiplicity n models the number of jobs that, at the same time, are temporarily blocked (if the failure is due to a machine on which n jobs are running, n jobs experience the failure).

Figure 4.3c models the **abort** of a job. Transition af models the abort of a running job; upon its firing a token is removed from place R and added to place A. The repair consists in the job re-enqueuing; this is modeled by transition ar which takes a token from A and places a token in Q. As discussed, aborts may be due to the same causes of permanently blocked jobs failures; hence, the occurrence of one inhibits the manifestation of the other. This is modeled by adding an inhibitor arc from place QF to transition af and another inhibitor arc from place A to transition qf.

Finally, the **exiting failure** is modeled by the SRN in Figure 4.3d. Transition ef models the failure in returning the results of a job by removing a token from C and adding, upon firing, a token to place EF. Also in this case, the repair consists in the re-enqueuing of the job. This is modeled by transition er, which removes a token from EF and adds a token to place Q.

Since a repair is considered for each type of failure and the corresponding Markov chain does not present an absorbing state, the proposed one is an availability model [128]. Also this model is repeated if different types of jobs runs in the system.

Inputs. This failure model requires as inputs the probability distributions of: (i) queue failures, (ii) running failures, (iii) aborts, and (iv) exiting failures.

Outputs. The output of the model is the availability of the system. We conceive the availability as the probability that a job is executed without experiencing any failures; in other words, it represents the chance that the result of a job is returned in the lowest possible time.

4.2.4 Consumability Model

The presented models cooperate to create a composite model for the joint analysis of performance, consumption, and dependability. Relations among performance model and failure model are easily identifiable, given the common places.

The power consumption of the system is computed using as number of running jobs of each type the number of tokens in the R places corresponding to the different types of jobs that run in the system. The evaluation of the power consumption due to failures depends on the type of the failure.

In the case of **queue and running failures**, the electric power waste can be quantified as the power absorption of the machines switched on uselessly. When a job is blocked (both permanently and temporarily) no dynamic power is consumed. Therefore, a token in QF or RF means that the static consumption of a certain number of worker nodes (the ones that cause the blocking of m and n jobs, respectively) is squandered. As an instance, if queue failures, on average, affect 16 jobs scheduled onto two nodes, while the jobs are blocked, the two nodes are idle and switched on in vain. Then, $W_s \cdot 2$ Watts are wasted while a token is in QF. The cost for running failures is estimated similarly.

In the case of **aborts**, the electric power used to execute the job before its abort is to be considered wasted. When a token is added in place A, for a period of time, they have been wasted (i) a portion of the static consumption of the node that was executing the job (other jobs running on the same machine do not fail, hence the remaining part of the static consumption is not squandered), and (ii) all the dynamic consumption due to the aborted

job. The static consumption of the node is the W_s of Equation 4.1 and is to be divided by the number of jobs that a node can execute (e.g., by 8 if each node has 8 CPU cores and it executes up to 8 jobs). The dynamic consumption of the job depends on its type k_i and it is the weight W_{k_i}. The period of time a job has been executed before its abort is the average time $E[t_{R_a}]$ a token is in place R before the firing of the transition af. This is given by:

$$E[t_{R_a}] = \frac{E[R] \cdot E[t_R]}{\lambda_a} \tag{4.2}$$

where $E[R]$ is the average number of tokens in place R, $E[t_R]$ is the average time a token is in place R and λ_a is the abort rate [128, 139]. $E[R]$ is estimated by the software tools commonly used for resolving the models [62] and λ_a is known (it is an input parameter of the model). The average time the tokens are in a place is to be estimated. To compute the mean time spent by tokens in a certain place, we adapt the *Little's formula* to the case of SRNs [128]. The formula states that the mean number of jobs in a queuing system in the steady state is given by the product of the arrival rate and the mean response time. If λ is the arrival rate and $E[t_R]$ is the mean response time, the mean number of running jobs can be estimated as $E[R] = \lambda \cdot E[t_R]$. Since λ is known, we can easily compute the mean duration of a job as $E[t_R] = E[R]/\lambda$.

If n types of jobs $K = \{k_1, k_2, \ldots, k_n\}$ are running in the system, each of them characterized by the arrival rate λ_{k_i} and with $E[R_{k_i}]$ running jobs, the average time a job of type k_i is running, that is the expected time a token is in place R_{k_i}, is given by:

$$E[t_R]_{k_i} = \frac{E[R_{k_i}]}{\lambda_{k_i}} \tag{4.3}$$

In the case of **exiting failures**, the electric power used for completing the job is assumed

as wasted. Hence, when a token is added to place EF, the consumption of a worker node for entirely executing the job is lost. This depends on the average duration of the jobs of a certain type, which is estimated as seen for the aborts.

Inputs. The consumability model requires as inputs: (i) the number of worker nodes N, (ii) the workload distributions (inter-arrival, scheduling, completion, exiting), (iii) the consumption weights W_s and W_{k_i} for each type of job $k_i \in K$ (in Watts), (iv) the failure distributions, and (v) the coverage and frequency of the fault tolerance mechanism (if implemented; see Section 4.3).

Outputs. The model provides as outputs: (i) the throughput of the system or the average time to complete the jobs as performance metrics, (ii) the total consumption of the system, (iii) the consumption due to failures, (iv) the consumption due to fault tolerance (if implemented), and (v) the availability of the system as the probability that a job is executed without experiencing any failures.

4.2.5 Applicability of the Model

The presented consumability model depicts the common behavior of a batch processing system involving its normal and faulty behavior, and consumption. The performance model is based on the common life cycle of jobs, the failures are the deviations from such life cycle for each of its stages, and the consumption is computed as a function of its load. The examples of failure occurrences validate the assumptions on the undesired events that can happen from the job execution until its termination, but they do not make those failures

specific to any systems. As a consequence, the model introduced in this dissertation can be adopted for the analysis of any batch processing systems.

Obviously, the transition distributions and the consumption weights, i.e. the model inputs, are specific to the system one want to study, as common to all evaluation approaches: each system has its own performance and usage features. As discussed in Section 3.2, workload related information (e.g., inter-arrival, scheduling, and completion distributions) can be estimated from the resource manager and scheduler logs, the weights of the consumption model can be computed from monitored consumption data, and failures related distributions can be estimated by means of log analysis. If more types of job are identified, the performance model can be replicated for each type, using the specific distributions. If more failure types are identified, since the modeled ones consider possible deviations during all the stages of the job life cycle, the rates can be summed or equivalent distributions computed [128].

These steps are illustrated in Chapters 5 and 6. The analysis can be performed for other systems as for the presented case studies, which exemplifies the steps of a systematic procedure for the consumability assessment.

4.3 Fault Tolerance Models

Several fault tolerance techniques can be adopted to mitigate the effects of the failures discussed in the previous Section 4.2.3. Job checkpointing and job replication are often employed in large scale processing systems [14, 24].

In the following, we model checkpointing and replication mechanisms with reference to the SRN of the batch system. This allows us to evaluate their cost in terms of energy consumption and to identify which one provides the best trade-off among performance, consumption and dependability.

4.3.1 Job Checkpointing

Job checkpointing consists in periodically saving a snapshot of a job state so that, in case of failure, its execution can restart from the last checkpoint, rather than from the beginning. It requires a machine to orchestrate the operations and another one to save the job checkpoints. We assume fixed checkpointing frequency (i.e. fixed interval of time between consecutive checkpoints) and coverage of the jobs (i.e. number of jobs whose state is to be saved and that can be recovered from a checkpoint if they fail).

The orchestration mechanism can be easily modeled by the SRN in Figure 4.4. $start_{CHK}$ models the start of the checkpointing operations, while $stop_{CHK}$ models the end of saving the job states. When performing checkpointing operations, another checkpoint cycle cannot start, as modeled by the inhibitor arc from place CHK to transition $start_{CHK}$.

Repairs after failures are to be modeled as well. Checkpointing allows the system to mitigate the effects of aborts and exiting failures. In the case of queue failures, since the execution of the blocked job is not started, no state can be saved. In the case of a running failure, the job may be still completed.

The SRN in Figure 4.5 models the abort and its repair when using checkpointing. After

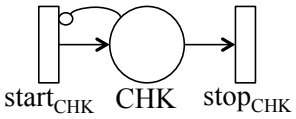


Figure 4.4: SRN modeling the checkpointing operations.

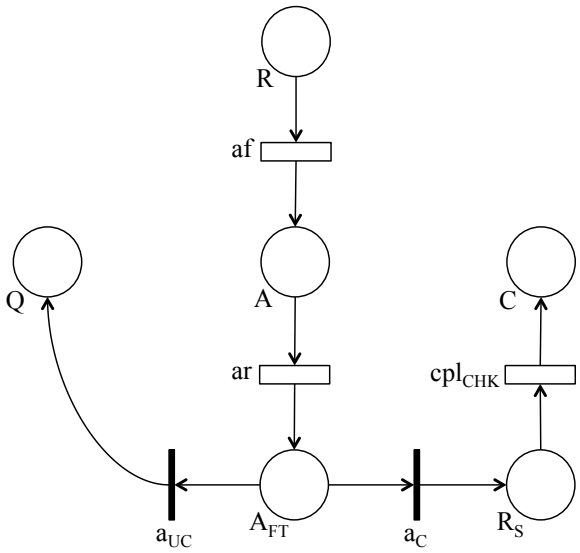


Figure 4.5: SRN modeling the abort when checkpointing is adopted.

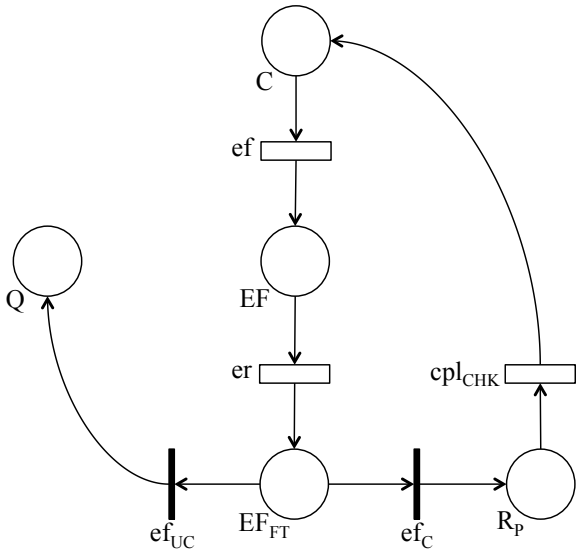


Figure 4.6: SRN modeling the exiting failure when checkpointing is adopted.

a job aborts, two cases are to be contemplated for its repair[2]: (i) with probability $1 - cov$ the failure is uncovered (immediate transition a_{UC}), that is the state of the job has not been saved; in this case, the job is to be re-submitted: a token is removed from place A_{FT} and added to place Q, as in the case of no fault tolerance; (ii) with probability cov (immediate transition a_C) the abort is covered, that is the system has a checkpoint of the job, which allows it to complete the execution of the job starting from the last checkpoint; thus, a token transits from A_{FT} to place R_S and completion is modeled by transition cpl_{CHK}, which, upon firing, removes a token from R_S and adds it to C. For the rate of transition cpl_{CHK} we consider the worst case, i.e. the failure happens in the instant of time exactly before the state of the job is saved; if the checkpoint is performed every n minutes, the last n minutes of execution are assumed to be re-executed.

The SRN modeling the exiting failure and its repair when using checkpointing is shown in Figure 4.6. Also in this case, covered and uncovered failures are to be contemplated: (i) with probability $1-cov$ (immediate transition ef_{UC}) the state of the job has not been saved; hence, the job is to be re-submitted: a token is removed from place EF_{FT} and added to place Q, as in the case of no fault tolerance; (ii) with probability cov (immediate transition ef_C), the system has a checkpoint of the job, which allows it to execute only the remaining part of the execution; in this case, a token transits from EF_{FT} to place R_P and completion is modeled by transition cpl_{CHK}, which, upon firing, removes a token from R_P and adds it to C. For the rate of cpl_{CHK} we do the same observations done for aborts.

[2]For additional information on failure coverage see Section 2.2.2.

4.3.2 Job Replication

Replication consists in running the same job twice on different machines. In this case, the coverage represents the portion of jobs for which a replica exists. If the coverage is $cov = 0.5$, half of the submitted jobs is replicated. To model job replication, we adapt the performance model to mimic the duplication of a certain number of jobs.

With reference to the performance model in Figure 4.2, (i) the multiplicity of the arc $arr \rightarrow Q$ is changed to consider the portion of replicated jobs (for each newly arrived job, $1+cov$ tokens are to be added in Q); (ii) even if replicated, the result of a job is to be returned once; hence, the multiplicity of the arc $C \rightarrow exg$ is also modified (for each completed job, $1 - cov/3$ tokens are removed from place C). Note that if $cov < 1$, multiplicity of these arcs is not an integer. In such a case, or fluid SRNs [62] have to be used or a lightly different model must be created to model replication.

Replication allows the system to mitigate the effects of queue failures, exiting failures, and aborts. Also the delay caused by running failures is masked by the chance that the job replica does not experience any failures.

In Figure 4.7, we show the SRN modeling the queue failure and its repair when replication is adopted. Differently from checkpointing, the user becomes aware of a job failure only if it is not replicated or both the job and its replica experience a failure. Once transition qf fires, a token is removed from Q and added to place QF_{FT}, then, two cases can take place: (i) with probability $1 - cov$ the failure is not covered (immediate transition qf_{UC}) and there is no replica of the failed job, hence it must be re-submitted, as in the case of no

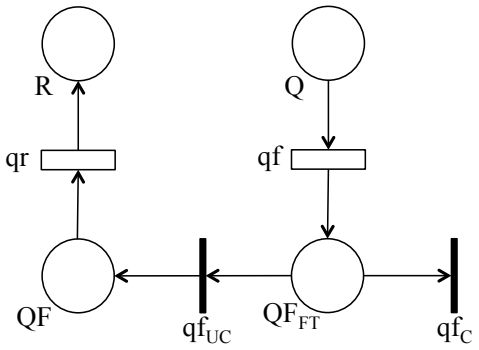


Figure 4.7: SRN modeling the queue failure when replication is adopted.

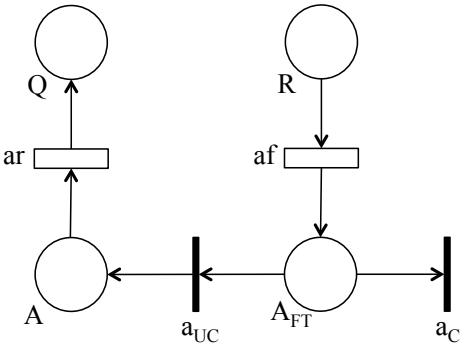


Figure 4.8: SRN modeling the abort when replication is adopted.

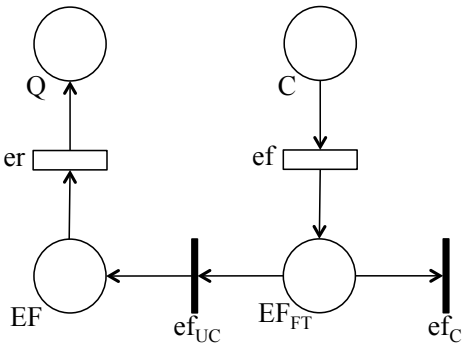


Figure 4.9: SRN modeling the exiting failure when replication is adopted.

fault tolerance; (ii) with probability *cov* the failure is covered (immediate transition qf_C), the job is replicated and the user may receive the result from the not failed replica of the job.

Aborts and exiting failures are modeled similarly, as depicted in Figures 4.8 and 4.9, respectively.

4.3.3 Consumption due to Fault Tolerance

The consumption due to the fault tolerance depends on the adopted technique.

Checkpointing. In the case of checkpointing two machines are used, one for the orchestration of the technique, the other one for storing the states of the jobs. The static component of the consumption is due to two additional switched on nodes in the system. The dynamic consumption of the two machines is present during the checkpointing operation (i.e., when, after a certain period of time, the checkpoint starts and states are saved).

Replication. In the case of replication, a machine is used for the orchestration and additional worker nodes are used for executing the replicated jobs. For the orchestration, the static consumption is considered, while the dynamic one is neglected, since due to the use of a machine for few seconds to start the execution of a job on two different worker nodes. For the replicated execution, both the static and the dynamic consumption of additional worker nodes are to be accounted for.

4.4 A Metric for Consumability

We defined the ***consumability*** as the ability of a processing system to produce useful work in a certain time with energy efficiency, that is to properly exploit the incoming electric power given the inevitable occurrence of failures. The aim is to have a unified measure of performance, consumption and dependability to evaluate the energy efficiency of processing systems.

As discussed in Section 1.3, energy efficiency is usually computed as the relation between a performance metric and a consumption metric. While useful in thermodynamics, where the amount of useful work produced is a function of the amount of high-temperature heat input, for processing systems, this ratio if very different from metrics commonly adopted in engineering and physics, where the energy efficiency is evaluated as the relation between the system's useful power output and the power input supplied to it:

$$\eta = \frac{\text{Useful power output}}{\text{Useful power input}} \tag{4.4}$$

The objective of an energy efficiency metric is to relate what the system produces to what it receives for allowing it to produce. It is usually dimensionless and can be expressed as a percentage.

Our aim is to have a metric that also encompasses the consumption due to failures (which includes the consumption due to fault tolerance, if implemented). The consumption due to failures corresponds to the fraction of the incoming power that is wasted by the

system and not transformed in useful work. Hence, *a processing system is **consumable** if the portion of incoming power wasted due to failures is small.* In fact, the properly exploited incoming power coincides with *how much the system produces* and the consumption due to failures covers with *how much it is dependable.* Since the energy consumed for having a fault tolerance mechanism contributes to make the system dependable, its cost is encompassed in the consumption due to failures.

Equation 4.4 is adapted to the case of processing systems by considering:

- **Useful power output**: the electric power actually used to provide results to users; it can be computed as the difference between the total consumption of the system and the consumption due to failures;
- **Provided power input**: the total power consumption of the system.

Formally, the consumability of a processing system is quantified as (i.e., the energy efficiency of a processing system when also failures are contemplated is):

$$\eta = \frac{W - W_f}{W} \tag{4.5}$$

where W is the total power consumption of the system and W_f is the power consumption due to system malfunction, which also includes the consumption for the fault tolerance technique, if any.

Since both the numerator and the denominator are in Watts, η is dimensionless. Since $W_f \geq 0$ and $W - W_f \leq W$, then $0 \leq \eta \leq 1$. This allows one to express the consumability of a processing system as a percentage.

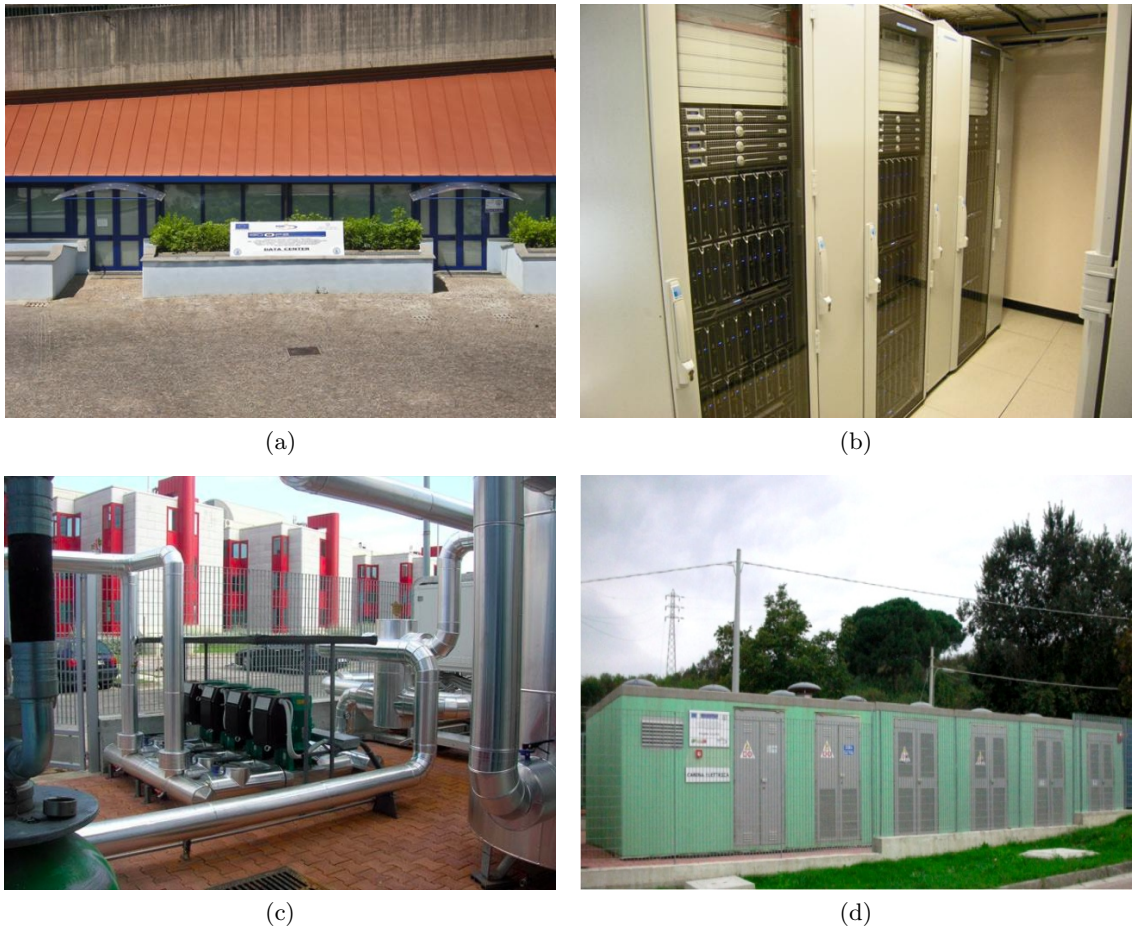
Chapter 5

Case Study 1: Consumability of a Scientific Data Center

In this chapter, it is discussed the consumability analysis of a real scientific data center at the Federico II University. To evaluate model inputs (i.e. transition distributions for performance and failures, and weights for the consumption) workload, consumption and failure data are analyzed. Outputs from the solved model reveal the energy consumption due to failures and the impact of implementing fault tolerance on throughput, power consumption, and availability. The consumability of the system under various configurations is evaluated with the proposed metric. It shows to be valuable to support the decisional process for balancing costs and benefits through the proper configuration of the system.

5.1 The S.Co.P.E. Data Center

Our experimentation is based on the analysis of the S.Co.P.E. scientific data center (Figure 5.1), on which we have direct management experience. S.Co.P.E. is a batch processing system at the Italian Institute of Nuclear Physics (INFN) and at the Federico II University of Naples used in the framework of a more general GRID infrastructure [129]. Specifically, S.Co.P.E. performs as a Tier-2 resource of the Worldwide LHC Computing Grid (WLCG) [19] and it is also used for other research activities of Federico II University. The acronym



(a) (b)

(c) (d)

Figure 5.1: The S.Co.P.E. data center: (a) external view, (b) some racks, (c) cooling system, (d) power plant.

stands for "Sistema Cooperativo per Elaborazioni Scientifiche Multidisciplinari", i.e. collaborative system for multidisciplinary scientific applications. The S.Co.P.E. data center includes a power plant (capable of delivering 1MW of electric power in continuous mode), a cooling system (capable of dissipating 2,000 W per cubic meter and 30,000 W per rack), a Gigabit Ethernet and Infiniband networking, NAS and SAN storage working with a Fibre Channel Infrastructure. It is composed of 512 Dell PowerEdge M1000e servers, each equipped with 2 quad core Intel Xeon CPUs, 16GB of memory (totaling 4096 cores with

2GB of memory each). Worker nodes are equipped with Scientific Linux. For jobs queueing and scheduling and resource management, S.Co.P.E. uses Maui/TORQUE [1].

As for the **fault tolerance**, the S.Co.P.E. data center has no mechanisms to cope with failures discussed in Section 4.2.3. Technicians manually solve failures due to blocked jobs. By querying the resource manager, they identify blocked jobs and the machines where such jobs have been scheduled. The reboot of such machines is usually enough to solve the problem. The check is performed every 4-5 days.

5.1.1 The Monitoring System

The data center is equipped with several sensors and tools for monitoring the system. Collected data are useful for evaluating the inputs for the consumability model, as presented in next Sections.

Data collectable through these monitoring systems can be classified in three broad categories: performance, environment, and operational data (see Section 3.2).

These data are monitored in the S.Co.P.E. system as follows.

Resources utilization. To monitor resources utilization, S.Co.P.E. adopts Ganglia, an open source monitoring system for clusters and grids, developed at the University of California at Berkeley [130]. It monitors the utilization of node resources. Exemplary parameters are *cpu*, which reports CPU usage, *cpu_wio*, which represents the time spent by the CPU in waiting for I/O, *RAM*, which indicates the used RAM. Data are stored using the RRDTool[1],

[1]Round-Robin Database Tool - http://oss.oetiker.ch/rrdtool/.

specifically designed to handle time-series data that are stored in a round-robin database to keep the required storage space constant over time and to simplify the visualization by means of graphs (usually trends with respect to time of CPU, memory, CPU waiting for I/O, ...).

Monitoring of S.Co.P.E. with Ganglia started in May 2009. However, the RRDTool changes stored data granularity in order to reduce allocation space. As an instance, data of last day are stored with 5 minutes granularity, data of last week are stored with 1 hour granularity, data of last month are stored with daily granularity.

Workload. Data about the workload of the system are collected using logs produced by the resource manager and JobMonArch[2], a plug-in for Ganglia, which provides batch job monitoring. Jobs and queues are monitored with information about jobs per node, job status, job resource requirements and use. Other data about jobs (useful for workload characterization) are the time when the job was submitted, the time when the job execution started, and the time when the job execution finished. Specifically, for each job, we have the following information:

- **Job ID** (J_{ID}): a unique identifier of the job in the system.
- **Queue** ($Queue$): the scheduling queue to which the job was submitted; jobs are allocated to different queues depending on their duration (e.g., *long* or *short* jobs), the specific organizations that use the system, and on the specific project they belong to; however, queue policies are not always respected.

[2]Job Monitoring and Archiving - http://sourceforge.net/projects/jobmonarch/

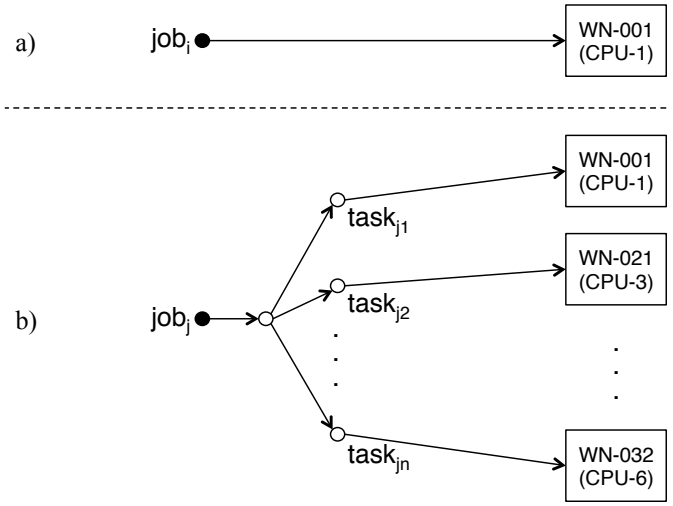


Figure 5.2: Scheduling (a) of a common job and (b) of a parallel job.

- **Arrival Time** (t_a): the time (in Unix timestamp) when the job was submitted to the system.
- **Start Time** (t_s): the time (in Unix timestamp) when the execution of the job started.
- **Completion Time** (t_c): the time (in Unix timestamp) when the execution of the job finished.
- **Node** (WN): the worker node of the system in which the job ran.
- **CPU utilization** (CPU): average usage of CPU by the job.

In the case of parallel jobs, the information is specialized *for each task*. Commonly, a job job_i is scheduled on a processor core of a worker node (Figure 5.2a); in the case of a parallel job job_j, it is made up of n tasks $task_{j_1}$, $task_{j_2}$, $\cdots$, $task_{j_n}$, scheduled on the processor cores of some worker nodes (Figure 5.2b). We have all the information as for non-parallel jobs for each task (e.g., arrival, scheduling, completion, resource usage).

Collection of workload related data started in May 2009.

Consumption. Chassis Management Controllers (CMCs) provided with Dell M1000e servers are used for the power monitoring. Each chassis contains 16 servers. CMCs are managed and monitored via SNMP. We created a Java application to collect power consumption data from the chassis and to store them in a MySQL database. Data are collected every 15 minutes. Power data collection started in December 2011.

Logs. Another useful source of information is the set of logs collected from the various software layers. We used operating system logs (*syslog*), and logs created by the resource manager and by the scheduler.

5.2 Data Analysis for Populating the Model

Inputs for the consumability model presented in Section 4.2 were evaluated through the analysis of data collected from the S.Co.P.E. data center. In the following subsections, we use:

- *Performance and workload data* (Section 5.2.1), to identify the types of jobs present in the system and to evaluate, for each type, inter-arrival, scheduling and completion distributions.
- *Power consumption data* (Section 5.2.2), to evaluate static consumption of the system and weights for the dynamic consumption.

- *Logs* (Section 5.2.3), produced by operating systems, scheduler and resource manager, for estimating the rates of the failures.

5.2.1 Workload Analysis

The analysis of the workload was done following the common steps as outlined in several papers by Calzarossa [16, 17, 18]. Namely, we first identified clusters of jobs for resource usage, then we characterized each cluster for inter-arrival, scheduling and completion distributions. Another approach could have been to cluster the jobs with respect to the needed distributions, that is, by considering inter-arrival, scheduling and completion times. However, this approach would have produced meaningless clusters from the resource utilization point of view and hence for consumption analysis - since the consumption strictly depends on the resource utilization.

Available data are related to 231,237 jobs executed on $N = 250$ nodes in a year (January 18th 2010 - January 17th 2011).

Types of jobs

Workload and resource utilization data at our disposal did not overlap for a long period. Hence, we used the information on the utilization of the CPU reported in the workload data for clustering the jobs.

We adopted the *k-means* method since more suitable than hierarchical clustering for

large amounts of data (more than 100,000 observations). The method selects a set of k points (cluster seeds) as a first guess of the means of the clusters, then assigns each other point to the nearest seed to form temporary clusters. The new mean is computed and new seeds determined. The process continues until seeds does not change anymore.

A good clustering requires the sums of the distances of the elements in a cluster from the seed (or centroid) being small, where *small* depends on the available data. For the distance, we used the squared Euclidean one. We looked at the trend of the *sums of point-to-centroid distances* searching the best trade-off between clustering error (given by the distances) and number of clusters (onto which depends the complexity of the model). The trend is depicted in Figure 5.3. For $k = 1$ the sums of point-to-centroid distances is obviously maximum. Just after the knee (number of clusters equals to 2), the curve is close to the asymptotic value. Hence, we selected $k = 3$ for which the sums of point-to-centroid distances is 785.63.

The three clusters have centroids 0.25, 0.63, and 0.96. We classified the jobs as I/O jobs, CPU jobs and HPC jobs as follows:

1. *I/O boud jobs*: $0 < CPU \leq 0.4405$;
2. *CPU boud jobs*: $0.4405 < CPU \leq 0.89$;
3. *HPC jobs*: $CPU > 0.89$

The classification covered the 100% of the jobs. 46,430 jobs were in the *I/O jobs* set, 162,491 in the *CPU jobs* set, and 22,316 in the *HPC jobs* set.

For corroborating the classification, we performed a clustering also on the reduced dataset containing other resource utilization indicators. We also included the Linux *wio*

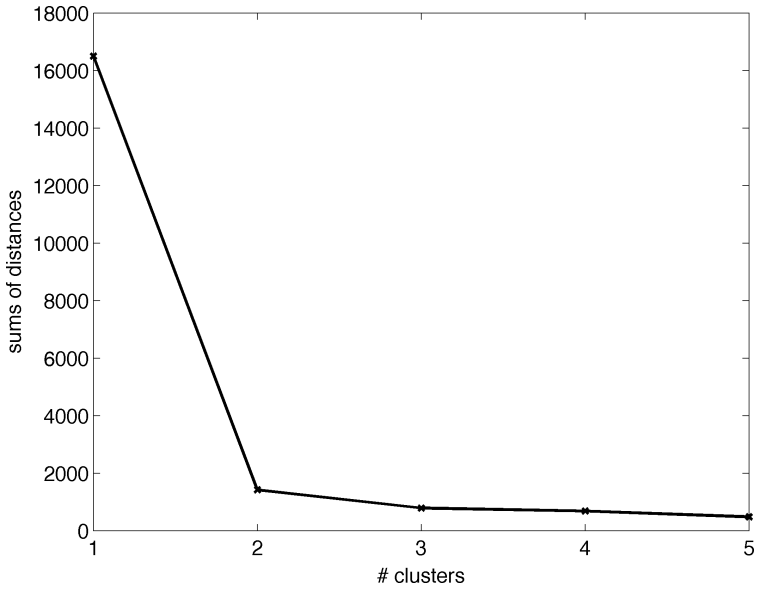


Figure 5.3: Trend of the sums of point-to-centroid distances when varying the number of clusters.

metric, which indicates the amount of time the CPU waits for I/O operations. The k-means for $k = 3$ suggested the following classification:

1. *I/O bound jobs* - $0 < CPU \leq 0.45$, $wio > 4.4$;

2. *CPU bound jobs* - $0.45 < CPU \leq 0.80$, $1.4 < wio \leq 4.4$;

3. *HPC jobs* - $CPU > 0.8$, $wio < 1.4$.

In this case the clustering classified the 91% of the jobs. An exemplary clustering on 24 hours data is reported in Figure 5.4.

With reference to the formalism introduced in Section 4.1, in the case of the S.Co.P.E. system, there are three types of jobs and $K = \{I/O, CPU, HPC\}$. The performance model of Section 4.2.1 is specialized for the identified job types as shown in Figure 5.5. We

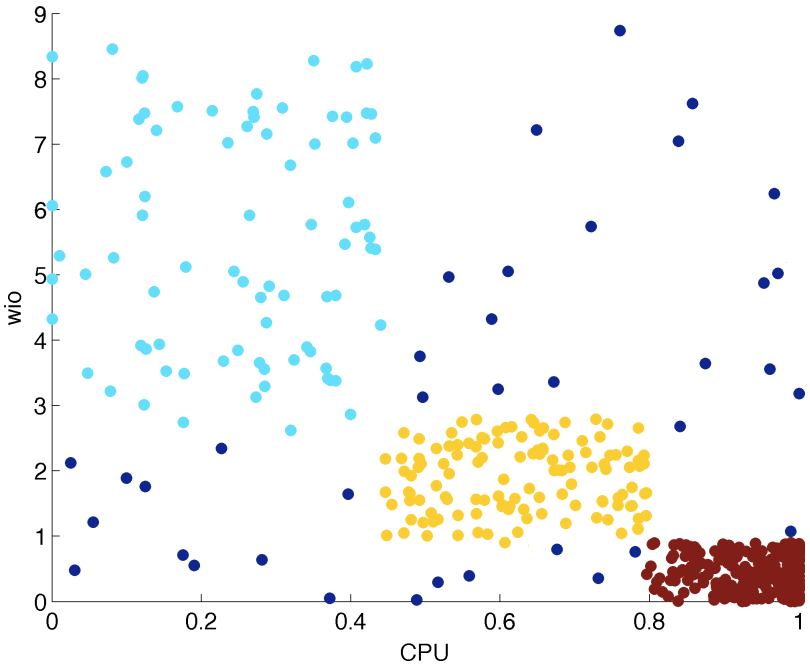


Figure 5.4: Clustering of the jobs with respect to CPU usage and *wio*.

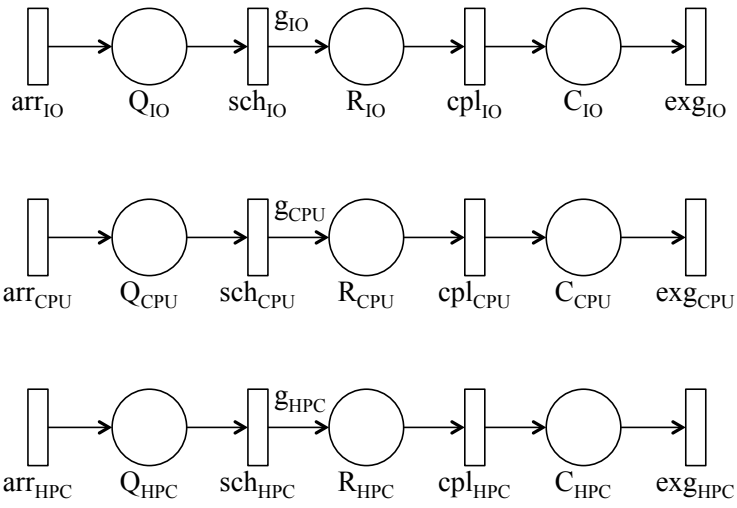


Figure 5.5: Performance model specific to the S.Co.P.E. data center.

also observed that each machine can execute up to 8 jobs, as are the cores of the CPUs. The execution capacity is modeled by the guard functions g_{IO}, g_{CPU} and g_{HPC}. The execution of a job starts if

$$(AR_{IO} + AR_{CPU} + R_{HPC}) < (N \times 8) \tag{5.1}$$

where AR_{k_i} is the the number of running jobs type k_i and N is the number of worker nodes.

Inter-arrival, scheduling and completion distributions

By using real data on submission (t_a), starting (t_s), and completion time (t_c) of the jobs of each cluster, it is possible to evaluate the transition distributions to be used in the model, for each job type. Inter-arrivals are computed as the difference between arrival times of two consecutive jobs ($\textit{inter-arrival}(j_i) = t_a(j_{i+1}) - t_a(j_i)$); scheduling times are computed as the difference between the start time and the arrival time ($\textit{scheduling}(j_i) = t_s(j_i) - t_a(j_i)$); completion times are computed as the difference between the completion time and the start time ($\textit{completion}(j_i) = t_c(j_i) - t_s(j_i)$).

To evaluate the distributions, we assumed all the times be independent and identically distributed (iid). In the case of parallel jobs, times related to all the tasks were grouped in one measure (by summing the times and dividing the sum by their number) to avoid correlated entries in the data set, similarly to what shown in [128].

We used the statistical software JMP[3] to analyze the data and to estimate the continuous distributions that best fits the data. Several outliers have been removed from the dataset

[3]http://www.jmp.com

since prevented to obtain a good fitting. Values were considered possible outliers only beyond 1.5 times the interquartile range (Q3 - Q1) [70].

Figure 5.6 illustrates the histogram of inter-arrival, scheduling and completion times (black-surrounded bars), the outlier box plot (in the top of each figure), and the distribution that best fits the data (red lines). For each distribution, parameters (unit of measurement is the second) and goodness of fit are reported.

As for goodness of fit, the JMP tool adopts EDF (Empirical Distribution Function) tests. For Exponential and LogNormal distributions, the goodness of fit is evaluated with the Kolmogorov's D test. In the case of the Gamma distribution for CPU jobs completion times, no goodness of fit test is provided (in this case, the diagnostic plot in Figure 5.7 is used).

The expressions of the probability distribution functions are the following:

$$Exponential(x;\sigma) = \frac{1}{\sigma} e^{\frac{-x}{\sigma}};\ x \geq 0;\ \sigma > 0 \tag{5.2}$$

$$LogNormal(x;\sigma,\mu) = \frac{1}{x\sqrt{2\pi\sigma^2}} e^{-\frac{(log(x)-\mu)^2}{2\sigma^2}};\ x \geq 0;\ \sigma > 0 \tag{5.3}$$

$$Gamma(x;\alpha,\sigma,\theta) = \frac{1}{\Gamma(\alpha)\sigma^{\alpha}} (x-\theta)^{\alpha-1} e^{\frac{-(x-\theta)}{\sigma}};\ \alpha,\sigma > 0 \tag{5.4}$$

The Kolmogorov test considers to test the hypothesis that X follows a specified distribution function: H_0 : for all t, $F_X(t) = F_0(t)$. Introducing the empirical CDF on a a random sample of size n:

$$\hat{F}_n(t) = \begin{cases} 0 & , t \leq t_1 \\ \frac{i}{n} & , t_i \leq t \leq t_{i+1} \\ 1 & , t_n \leq t \end{cases} \tag{5.5}$$

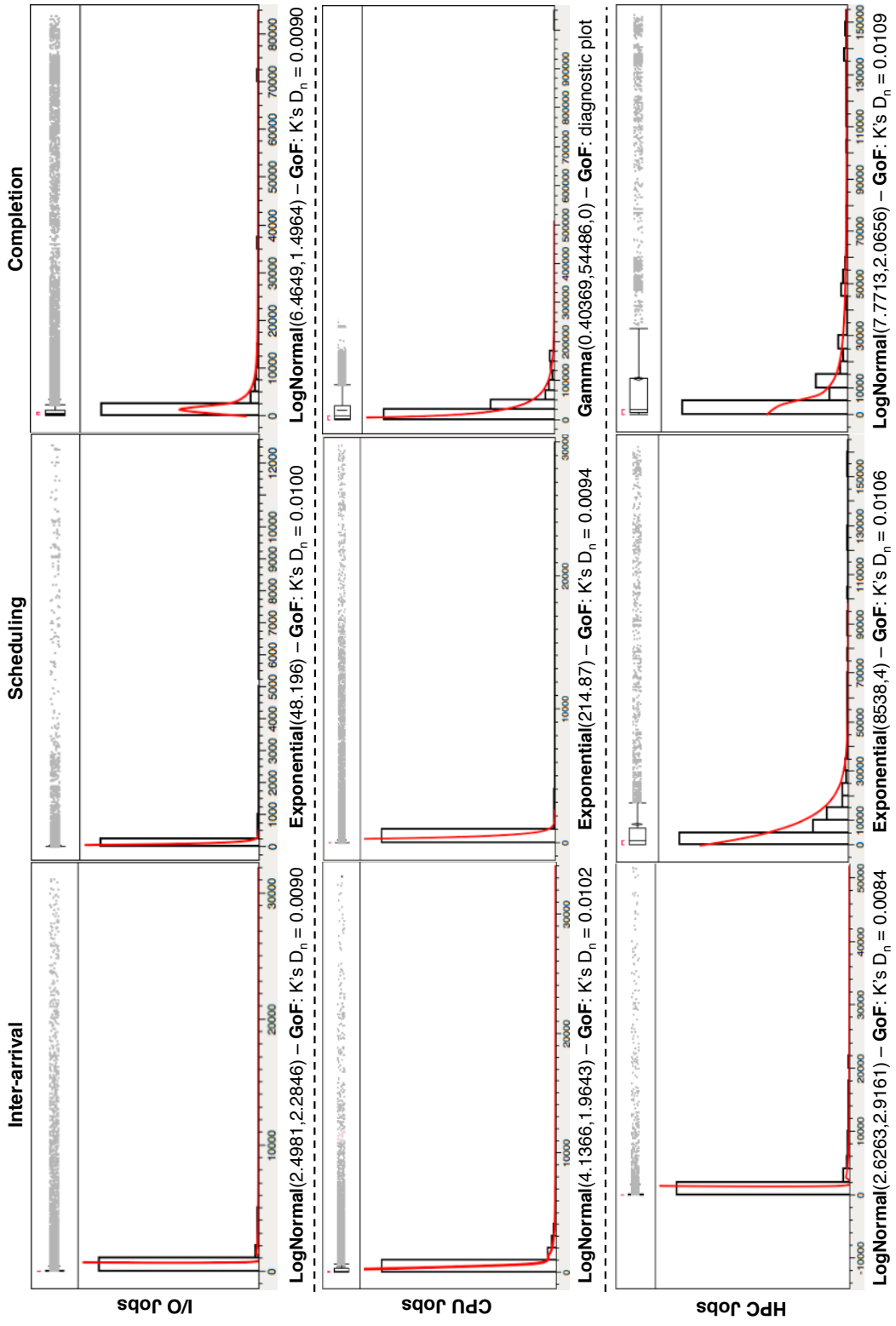


Figure 5.6: Histograms and distributions of arrival, scheduling and completion times for the three types of jobs.

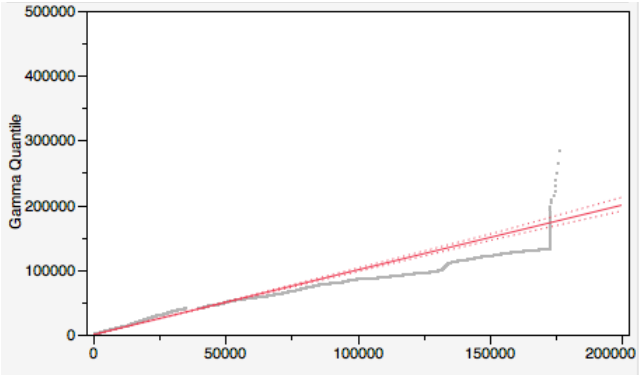


Figure 5.7: Diagnostic plot for the completion time distribution of CPU jobs.

the D_n statistic is defined as the maximum deviation of $\hat{F}_n(t)$ from the theoretical CDF $F_0(t)$:

$$D_n = \sup_t |\hat{F}_n(t) - F_0(t)| \tag{5.6}$$

The null hypothesis H_0 can not be rejected at a level of significance α if the observed value of D_n is less than a critical value $d_{n;\alpha}$. We use a level of significance of 95%.

I/O jobs. We started from analyzing the 46,430 jobs in the I/O cluster to estimate the probability distributions of inter-arrival times, scheduling times and completion times. This cluster presented the more regular behavior (most of the jobs falling in this cluster are related to the same experiment). For analyzing the inter-arrival times, 1,906 outliers were eliminated through the outlier box plot; 210 outliers were eliminated for estimating the scheduling distribution; 1,960 outliers were eliminated from the completion times. For all the fitting distributions, $D_n \leq D = 0.0100$; hence, the null hypothesis can not be rejected at the 95% level of significance.

CPU jobs. In the case of the 162,491 CPU jobs we removed 9,112 outliers for estimating the inter-arrival times' distribution, 892 outliers were eliminated for the scheduling distribution, and 1,205 outliers were eliminated from the completion times. In this case, the goodness of fit test for the inter-arrival distribution rejects the null hypothesis returning $D_n = 0.0102 > 0.0100$. However, no better distribution was estimated; moreover, the maximum deviation is only few larger than the critical value.

HPC jobs. Among the 22,316 jobs in the HPC cluster, 1,008 outliers were removed from inter-arrival times, 192 outliers from scheduling times, and 131 outliers from the completion times. In this case, the D_n for the scheduling and completion distribution is a bit larger than the critical value 0.0100.

Exiting distribution

Finally, for the exiting transition, we assumed it firing with a rate of one every 30 seconds. Thus, for all the job types, that transition has an exponential distribution with $\sigma = 30$ seconds.

5.2.2 Consumption Analysis

The consumption model of Equation 4.1 is specialized for the three types of jobs of S.Co.P.E.:

$$W_{tot} = W_s \cdot N + W_{IO} \cdot AR_{IO} + W_{CPU} \cdot AR_{CPU} + W_{HPC} \cdot AR_{HPC} \tag{5.7}$$

where N is the number of machines for which we have data (250), AR_{IO}, AR_{CPU} and AR_{HPC} the number of running I/O jobs, CPU jobs and HPC tasks, respectively. The data

Table 5.1: Results of the regression for the consumption.

Parameter	**Value [W] 95% c.i.**	***t* Ratio**	**Prob> |*t*|**	R^2	***F* Ratio**	***Prob>F***
W_s (intercept)	156.91 [148.28; 165.54]	354.53	<0.0001	0.71	6623.48	<0.0001
W_{IO}	4.09 [3.52; 4.66]	1.94	<0.0001			
W_{CPU}	10.47 [11.41; 9.53]	5.35	<0.0001			
W_{HPC}	13.58 [12.36; 14.80]	11.57	<0.0001			

analysis is performed to evaluate the weights W_s, W_{IO}, W_{CPU} and W_{HPC}.

We observe that the total consumption of the system can be seen as the response variable of a regression model whose predictor variables are the number of running jobs of each type [69]. The data collected from power distribution units and the workload include information about the total consumption of the system (W_{tot}) and the number of running jobs of each type (AR_{IO}, AR_{CPU},AR_{HPC}). By means of multiple regression we can obtain the values of the weights (W_{IO}, W_{CPU}, W_{HPC}).

We adopted multiple linear regression based on least squares (Standard Least Squares model - used for a continuous-response fit to a linear model of factors).

First we considered all the crossings of predictor variables (AR_{IO}, AR_{CPU}, AR_{HPC}, AR_{IO}*AR_{CPU}, AR_{IO}*AR_{HPC}, AR_{CPU}*AR_{HPC}, AR_{IO}*AR_{CPU}*AR_{HPC}). Crossings appeared to not be statistically significant, therefore, we repeated the regression analysis without them. The results are summarized in Table 5.1 for N=250 worker nodes, using different tests for the goodness of the regression.

$\boldsymbol{R^2}$ is the square of the correlation between the actual and predicted response and

estimates the proportion of the variation in the response that can be attributed to terms in the model (i.e., that the model can explain). If $R^2 = 1$, there is a perfect fit (the errors are all zero). If $R^2 = 0$, the fit predicts the response no better than the overall response mean.

***F* Ratio** tests the hypothesis that all the predictor variable's coefficients (except the intercept) are zero. The F-test is used to check if the SS_{reg}, given by the difference between the total sum of squares (SS_{tot}) and the sum of squared error (SS_{err}), is significantly higher than the SS_{err}. The ratio $(SS_{reg}/v_{reg})/(SS_{err}/v_{err})$, where v_{reg} and v_{err} are degrees of freedom for SS_{reg} and SS_{err}, respectively, has an F distribution. If the computed ratio is greater than the value read from the F-table, the predictor variables are assumed to explain a significant fraction of the response variation.

Prob>F is the probability of obtaining a greater F-value by chance alone if the specified model fits no better than the overall response mean. Significance probabilities of 5% or less are often considered evidence that there is at least one significant regression factor in the model.

***t* Ratio** is the ratio of the estimate to its standard error and has a Student's t-distribution under the hypothesis that the true parameter is zero.

Prob> $|t|$ is the probability of getting an even greater t-statistic (in absolute value), given the hypothesis that the parameter is zero. Significance probabilities of 5% or less are often considered evidence that the parameter is not zero.

The consumption model is statistically significant both for the F-test and for the t-test. It is worth noting that the t-test specify that with 99% confidence each predictor is statistically different from zero given that the rest of the coefficients are in the model. The

Table 5.2: Transition distributions of the failure model.

Transition	**Distribution**
qf	Exponential(0.00130)
qr	Exponential(0.01096)
rf	Exponential(0.00158)
rr	Exponential(3.0176)
af	Exponential(0.00448)
ar	Exponential(0.1)
ef	Exponential(0.01950)
er	Exponential(0.1)

analytical expression of the consumption model for S.Co.P.E. is (weights are in Watts):

$$\begin{aligned} W_{tot} = \quad & 156.91 \cdot N + \\ & 4.09 \cdot AR_{IO} + \\ & 10.47 \cdot AR_{CPU} + \\ & 13.58 \cdot AR_{HPC} \end{aligned} \tag{5.8}$$

5.2.3 Failure Analysis

To estimate the inputs related to failures, we analyzed the log files produced by the operating system, by the scheduler, and by the resource manager.

Distributions used in the model are summarized in Table 5.2. Parameters are in hour^{-1}. Details about their evaluation are discussed in the following subsections.

Queue and running failures

For this analysis, we used data from four months of logs produced by the different software layers of the system. As discussed in Section 4.2.3, some jobs, after being queued, become permanently blocked (*queue failure*), while other jobs are temporarily blocked while running (*running failure*). The technicians of the data center, with a certain frequency, check the status of the jobs. When blocked jobs are found, worker node onto which they had been scheduled are rebooted, so solving the issue.

We found 5 occurrences of **queue failures**, each affecting, on average, 15.8 jobs. Given the small size of this sample dataset, we started with assuming an exponential distribution for those failures and computed the failure rate as the inverse of the mean time to failure. We estimated the rate of the queue failure to be $\lambda_{qf} = 0.00130$ (unit of measurement is hour^{-1}). We checked the goodness of fitting data with an exponential distribution through the Kolmogorov's D test. We calculated $D_n = 0.1678$. The critical value of D at 95% significance level for the 4 time to failures computed by the 5 occurrences is $D = 0.6239$ [128]. Since $D_n < D$, the null hypothesis that the exponential distribution fits the data can not be rejected at 95% level of significance.

As for the **running failures**, we found 6 occurrences of temporarily blocked jobs, each affecting, on average, 30.4 jobs. We estimated the rate of the running failure $\lambda_{rf} = 0.00158$ (unit of measurement is hour^{-1}). To test the goodness of fit, we calculated $D_n = 0.2241$. The critical value of D at 95% level of significance for 5 samples (inter-arrivals computed by the 6 failure occurrences) is $D = 0.5633$. Since $D_n < D$, the exponential distribution

fits the data at 95% level of significance.

For the arc multiplicities m and n of the failure model in Figure 4.3, we assumed 16 and 30, respectively. As a matter of fact, we had to select integer values for the model. Hence, we selected the ones closer to the average numbers of affected jobs.

As for the repairs, we considered when the job passed from the blocked status to the running status (due either to the continuous status change or to the reboot of the node).

In the case of **queue failure repair**, we found data on 63 jobs and estimated the repair rate being $\mu_{qr} = 0.01096$ (hour^{-1}). To test the goodness of fit of the exponential distribution with rate μ_{qr}, we computed $D_n = 0.1492$. For 63 samples at 95% significance level, $D = 0.1713$. Since $D_n < D$, the exponential distribution fits the data at 95% level of significance.

For the **running failure repair**, from data on 152 jobs, we estimated the repair rate $\mu_{rr} = 3.0176$ (hour^{-1}). In this case, $D_n = 0.1042$ and $D = 0.1103$ at 95% significance level.

Aborts

As discussed in Section 4.2.3, aborts may be due to the same causes of queue failures; hence, the rate estimated for queue failures is a component of the abort failure rate. However, we found also other causes for aborts.

We used log analysis techniques presented in Section 3.2 for the system and scheduler logs. Entries of the logs were filtered with keywords such as "[job was/will be] killed", "unexpected [job termination]", "error", "failure", "alert". 196,497 interesting entries were

found over about 22GB of data from four months of logging. Entries related to failures caused by the users were filtered (examples are *segmentation faults* and *page allocation failures* due to user programs).

Some log messages reported about killed jobs ("LOG_ERROR::node_bailout [...] job will be killed").

Since some log messages may be representative of one single failure events, coalescence was adopted. Specifically, we used tupling (also called *temporal coalescence*) [35]. The technique groups in a tuple T_i all the events X in a certain temporal window W, by applying the rule

$$\textit{if } t(X_{i+1}) - t(X_i) < W \textit{ then add } X_{i+1} \textit{ to } T_i \tag{5.9}$$

where X_i and X_{i+1} are two consecutive messages and $t(\cdot)$ is the timestamp of the message. To select the value of W, we used the *knee rule*: given the curve representing the trend of the number of tuples corresponding to different values of W, the rule suggests to use the size of the window corresponding to points just after the knee of the curve. This tuples-per-window graph is reported in Figure 5.8. We found 8 tuples when using a window of 2000 seconds, which means that 8 distinct failures happened in the observation period. For those failures, the estimated abort rate is $\lambda_{af_{logs}} = 0.00318$ (hour^{-1}).

To test the goodness of fit of the exponential distribution with rate $\lambda_{af_{logs}}$, we computed $D_n = 0.1278$. The critical value for the 7 inter-arrivals of the 8 failures at 95% level of significance is $D = 0.4834$. Since $D_n < D$, the null hypothesis that the exponential distribution fits the data can not be rejected at 95% level of significance.

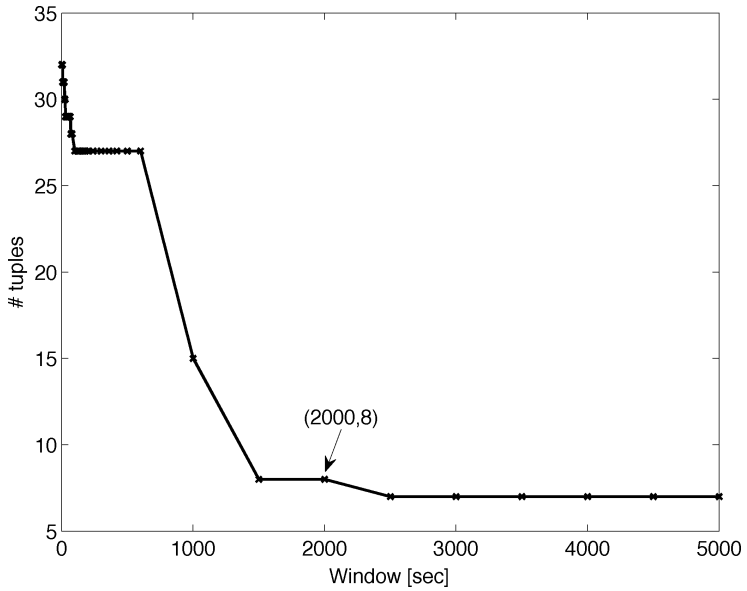


Figure 5.8: Number of tuples for different values of the time window.

By summing the value of $\lambda_{af_{logs}}$ to the rate of queue failures, we computed the abort failure rate $\lambda_{af} = 0.00448$ (hour^{-1}).

Exiting failures

Among the entries of the logs, some revealed issues in returning the results to final users ("LOG_ERROR::sys_copy [...] Output failed"). We classified those failures as exiting. In this case, just the time coalescence as used for aborts was not successful, since the relation among entries related to the same failure is not only due to time. Several entries are created for the same failure in different times and on different machines (e.g., several attempts to perform the operation, logs of the scheduler and of the worker node). Such messages have many common parts, such as the user name, the job name, the path where the result is to be copied. To figure out when messages belong to the same failure and group them, we

introduced an approach based on the distance between strings. Specifically, we used the Levenshtein distance [63]. It is a metric for measuring the distance between two strings as the minimum number of edits required to change one into the other. Formally, given two words x and y, the Levenshtein distance $L_\delta(x, y)$ between them is defined as :

$$L_\delta(x, y) := |x| + |y| - 2\rho(x, y) \tag{5.10}$$

where $|x|$ is the length of the word x, $|y|$ the length of the word y, and $\rho(x, y)$ the length of a largest common subsequence of x and y:

$$\rho(x, y) := max\{|w| \mid w \text{ is a common subsequence of } x \text{ and } y\} \tag{5.11}$$

It can be demonstrated that $L_\delta(x, y)$ corresponds to the minimum number of deletions and insertions needed to transform x into y.

For tupling the messages (this form of tupling can be classified as *spatial coalescence*), we group in a tuple T_i all the events X whose distance among them is less than a a certain value Δ, by applying the rule

$$if\ L_\delta(X_i, X_{i+1}) < \Delta\ then\ add\ X_{i+1}\ to\ T_i \tag{5.12}$$

where X_i and X_{i+1} are two consecutive messages and $L_\delta(\cdot)$ is the Levenshtein distance function applied to two messages of the log.

To select the maximum distance Δ by which two messages can be assumed to be related to the same failure, we computed the number of tuples returned for several values of the distance Δ and applied the knee rule. The tuple-per-distance trend is reported in Figure 5.9.

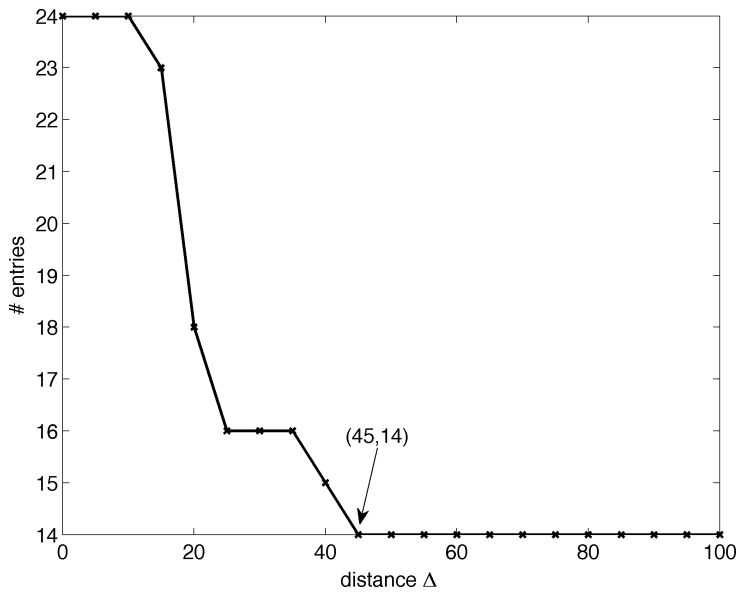


Figure 5.9: Number of tuples for different values of the Levenshtein distance.

In this case, the knee of the curve corresponds to $\Delta = 45$, for which 14 tuples are returned. 14 is also the asymptotic value of this curve. As a consequence, we assumed that in the observation period 14 exiting failures happened. We estimated the exiting failure rate $\lambda_{ef} = 0.01950$ (hour^{-1}).

For the Kolmogorov test of the exponential distribution with rate λ_{ef}, we computed $D_n = 0.1032$. For the 13 inter-arrivals, the critical value at 95% level of significance is $D = 0.3614$. Since $D_n < D$, the null hypothesis that the exponential distribution fits the data can not be rejected at 95% level of significance.

For aborts and exiting failures we assumed the jobs being resubmitted, on average, 10 hours after the failure. Hence the repair rates $\mu_{ar} = 0.1$ and $\mu_{er} = 0.1$ (hour^{-1}).

5.2.4 Fault Tolerance

For the distributions of checkpointing and replication models, we referred to the literature, since not having available data. As a matter of fact, the aim is to estimate the cost-benefit relation of implementing the fault tolerance. A detector of blocked jobs was implemented, instead.

Checkpointing

We assumed to perform a checkpoint every hour; for its duration, we assumed that 5 seconds were necessary for checkpointing each job and neglect network latency, as in [24]. The portion of jobs whose state is to be saved represents the coverage of the strategy. We assumed $cov = 0.5$.

Consumption. As for the consumption, we assumed that when a machine is used for performing checkpointing operations, all the resources (e.g., CPU) are used and its dynamic power consumption is the same of using all the available CPU cores for running HPC tasks (13.58×8 Watts). The static consumption is due to two additional machines switched on for performing checkpointing tasks (156.91×2 Watts).

Replication

In this case, the coverage represents the portion of jobs for which a replica exists. We assumed $cov = 0.5$. That is, half of submitted jobs is replicated.

Consumption. The dynamic consumption of the machine used for orchestrating the replication is neglected, since due to the use of a machine for few seconds to start the execution of a job on two different worker nodes. The static consumption is the one of a switched on node (156.91 Watts). For replicated jobs, both the static and the dynamic consumption are to be accounted. They are due to the additional number of running jobs, which is reflected by the additional number of tokens in the R places of the SRN.

The NOMAD detector

The blocking of the jobs often appears, causing both queue and running failures with a certain frequency (see Table 5.2). As discussed in Section 4.2.3, some jobs, after being queued, become permanently blocked (*queue failure*), while other jobs are temporarily blocked while running (*running failure*). The technicians of the data center use to solve issues related to blocked jobs by rebooting the nodes which they are scheduled on. Since performed by hand, these operations are not so frequent and well planned (in the case of the S.Co.P.E. data center, every 4-5 days). The idea was to automatize operations performed by the technicians to (i) detect blocked jobs and (ii) reboot nodes causing jobs to be in the blocked status. However, not all the jobs in the blocked status are permanently blocked; in some cases, a blocked job may become running again. That is, temporarily and permanently blocked jobs are to be distinguished, as well as queue and running failures.

We implemented a tool for detecting blocked jobs named NOMAD, which periodically

queries the resource manager (in the case of S.Co.P.E., Torque by means of the *qstat* command) to discover possible jobs in the blocked status (*WAITING* status, in the case of Torque). If a job is found to be blocked for two consecutive queries q_1 and q_2, it is marked as permanently blocked. If a job if found to be blocked in the last query q_3 and two queries before q_1, but not in the previous one q_2, it is marked as temporarily blocked.

NOMAD also performs recovery operations by rebooting the nodes causing permanently blocked jobs. At this aim, the tool queries the PBS scheduler (by means of a python script interacting with the PBS_Python Library[4]) and identifies the nodes where the jobs have been scheduled. Such nodes are rebooted, so mimicking the manual operations of the technicians. It is worth noting that these jobs become blocked while queued, hence the reboot does not cause their abort, but allows the nodes to start their execution, once the reboot completes.

The detection mechanism can be simply modeled by an SRN as the one discussed for the checkpointing (see Section 4.3.1, Figure 4.4). In this case, one machine is adopted. It queries the resource manager, and transition *start* represent the performing of such a query. Transition *stop* models the termination of the monitoring. We assume to perform the monitoring every 30 minutes and that it lasts 5 minutes.

The advantage of the automatic detector consists in reducing the time to repair, which, without any strategies, depends on the by-hand checking performed by technicians. Repair rates are significantly reduced when the failure is detected, since, in the worst case, it is solved in 30 minutes (interval between successive monitoring operations) plus 5 minutes

[4]http://freecode.com/projects/pbs_python.

(assumed average time to reboot a machine).

Consumption. The NOMAD detector implies the presence of an additional node. The static consumption is the one of a switched on node (156.91 Watts). As for the dynamic component, we assumed that when a machine performs the detection, all the available CPU cores are used as when running HPC tasks (13.58×8 Watts).

5.3 Model Validation

Validation is a main step for building a model providing trustworthy results. For validating the proposed consumability model, we used data from the S.Co.P.E. data center and threefold cross validation [138]. Specifically, we split data into three sets randomly selecting entries from the whole data set. Two of them were used to compute distributions and weights for the model (*training set*), the third for assessing the behavior of the models (*test set*). We compared the trends of the job completion and of the power consumption of the real system with the ones provided by the model. The number of failures is not checked separately since the samples from the real observations are already affected by the possible occurrence of failures.

Figure 5.10 shows the comparison of real data about the throughput from 100 hours of monitoring (circles) and the trend as captured by the overall model (dots), hence including also the failures that can happen. The throughput computed from the real data has more fluctuations with respect to the one provided by the model, which is able to describe the average behavior of the system anyway. A similar result is achieved for the consumption,

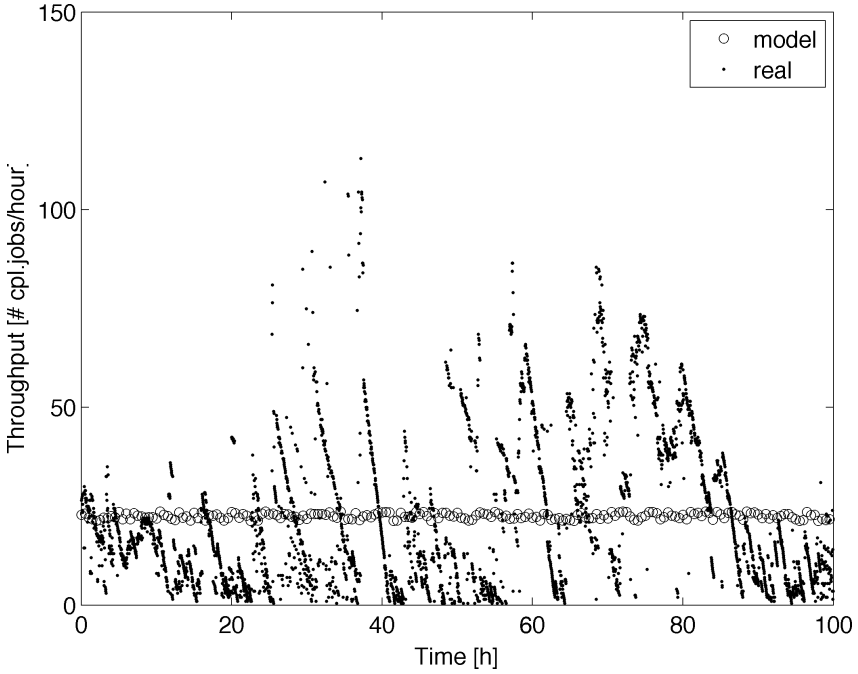


Figure 5.10: Comparison of the throughput of the real system (dots) with the results of the simulations of the model (circles).

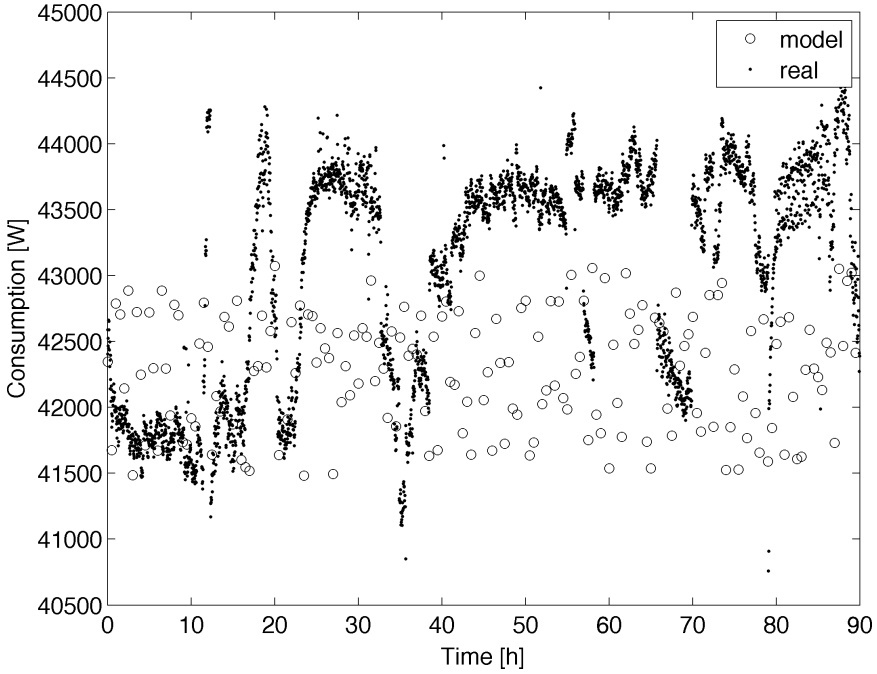


Figure 5.11: Comparison of the power consumption of the real system (dots) with the results of the simulations of the model (circles).

as shown in Figure 5.11, where the real data about the consumption (circles) and the consumption estimated by the model (dots) are compared.

5.4 Experimental Results

The model was solved by means of simulation (for the reasons explained in Section 3.4). Simulation was performed using SPNP [62], and results were computed with 95% confidence interval and maximum 5% half-width error. For transient analysis, *independent replications* simulation technique was used, with each simulation relative to 2 years of activity ($17,520$ hours). For steady state analysis ($t \to \infty$) the *method of batch means* was applied [128].

To check if two results (expected values) $\bar{x}$ and $\bar{y}$ are significantly different, the ***t*-test for the mean difference** was adopted [69]. We adopted this procedure since the observations are unpaired: there is no correspondence between the i-th value for achieving the expectation $\bar{x}$ and the i-th value for the expectation $\bar{y}$, since results of runs belonging to different simulations. The procedure considers the difference of the expected values, the standard deviations of such a difference and its number of degrees of freedom. Then, once the confidence interval for the difference is computed, if it includes the zero the difference is not significant (at a certain confidence level).

Results discussed in this Section are summarized in Table 5.3. First column reports the output of the simulation (expected values) for the ideal system without failures; the second

Table 5.3: Summary of the results for the consumability analysis of the S.Co.P.E. data center.

Metric	**Ideal system**	**Real system**	**Check.**	**Repl.**	**NOMAD**	**Check. & NOMAD**
Throughput [job/h]	35.071	34.750	35.002	35.046	34.891	35.069
Total Consumption [W]	42,484	42,279	42,579	43,353	42,326	42,536
Consumption due to Failures [W]	0	3,447.3	1,419.5	420.00	768.31	168.15
Consumption due to F. Tolerance [W]	0	0	157.45	2,400.4	72.862	165.92
Total Consumption due to Failures [W]	0	3,447.3	1,576.9	2,820.4	841.17	334.07
s.s. Availability	1	0.99221	0.99541	0.99927	0.99637	0.99985

column shows the results for the model of the real system (with failures, but without fault tolerance); other columns report the results for for the system with different fault tolerance techniques: checkpointing (third column), replication (fourth column), NOMAD (fifth column), and checkpointing and NOMAD together (sixth column).

5.4.1 Performance

Results on performance show that, *for a not saturated system, a performance metric is not able to identify the presence of failures.* To evaluate the performance of the system we employed the throughput as the expected number of jobs completed in the time unit (one hour, in our case), as commonly done for batch systems [94, 100].

The expected instantaneous throughput of the system for various fault tolerance strategies is reported in the second row of Table 5.3. It presents the largest variation when comparing the throughput of the ideal system with the one of the real system model (with failures, but without fault tolerance). In fact, the ideal system presents an expected number of completed jobs in one hour of 35.071 (jobs/h); when also failures are present, the system is able to complete 34.750 jobs every hour, i.e. the expected throughput decreases of just 0.9% (moreover, the t-test on unpaired observations revealed that confidence intervals for the differences include the zero; that is, the the performance is the same).

This behavior is due to the load that does not saturate the system, which always has available machines to execute newly arrived jobs, also if other ones have failed. As a consequence, after a certain time, the system is able to provide a throughput that is close

to the arrival rate of the jobs. Achieving a completion rate equals to the arrival rate in statistical equilibrium is a well known behavior of queuing systems [11]. We highlight that this result was estimated for the real workload of S.Co.P.E. Also other analysis present in the literature confirms that large processing systems are seldom near maximum utilization [132, 113].

The performance trend refutes all such metrics and benchmarks based on the performance-consumption relation for evaluating the efficiency of the system. A pure performance metric, as the ones discussed in Section 2.1.1, is not able to bring information on failures; therefore, the common approaches just evaluate the efficiency of an ideal system without malfunction.

5.4.2 Consumption

The consumption due to failures represents about 9% *of the total electric power required by the system*, as it is shown in Figure 5.12, which compares the consumption of the data center with the portion wasted due to malfunction.

The solution of the **real system** model showed that S.Co.P.E. is expected to consume 42,285 W, of which 3,560.9 W are due to failures. Assuming electricity cost being €0.43 per kWh[5], about €160,000 per year are spent for keep 250 worker nodes running; of this cost, €13,500 are payed for failures. Also, this cost only includes the energy directly required by the nodes. It neglects, for instance, the cooling system, which is commonly expensive and whose consumption is certainly affected by failures. If a job runs for 5 hours and then fails, it in vain produces heat, which is to be removed by the cooling system that consumes

[5]Electricity price for energy-intensive industries in Italy (http://www.enel.it).

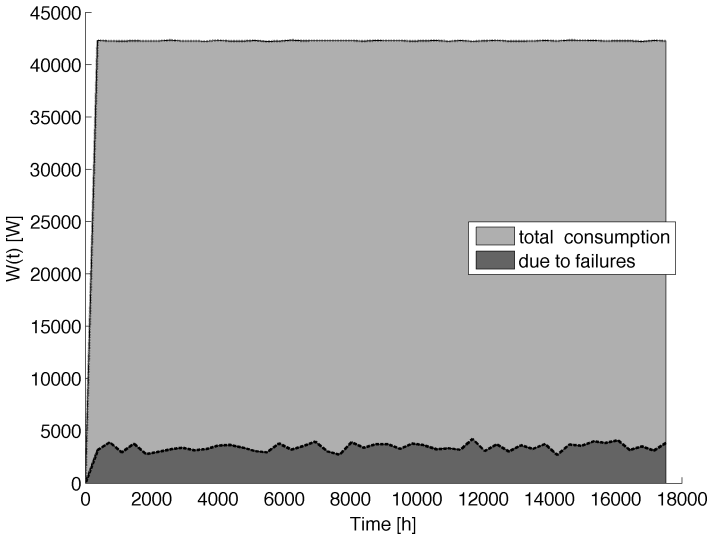


Figure 5.12: Expected instantaneous power consumption of the systems and portion due to failures.

energy. It is easy to imagine the cost for large data centers with tens or even hundreds of thousands of servers.

It is also worth noting that some failures cause the decrease of the overall consumption. As an instance, the consumption of the real system with failures is less than the consumption of the **ideal system** (see Table 5.3). This is due to idle machines causing queue and running failures (since appear idle), or crashed machines (since switched off). Hence, also *the consumption may be misleading for analyzing the actual behavior of a processing system and quantifying the cost of malfunction.*

These results for performance and consumption point out that energy efficiency metrics based only on their relation can barely measure the (faulty) behavior of real processing systems or the impact of fault tolerance and management strategies. This is demonstrated in detail in Section 5.5.

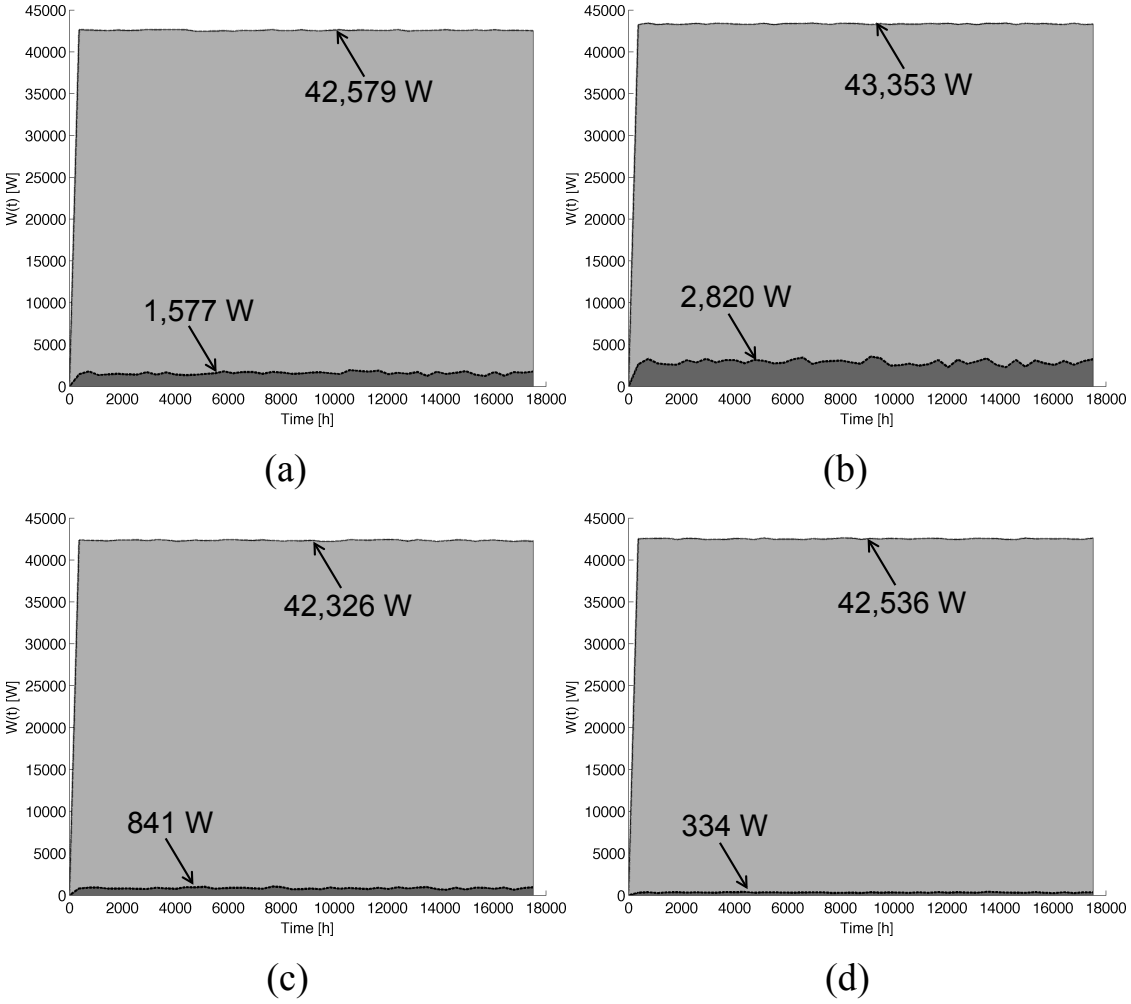


Figure 5.13: Expected instantaneous power consumption of the systems and portion due to failures when adopting (a) checkpointing, (b) replication, (c) NOMAD and (d) checkpointing and NOMAD together.

We also estimated the cost of the discussed fault tolerance techniques and evaluated the cost of the system malfunction as the sum of the power consumption necessary for executing the fault tolerance strategy and the consumption due to the occurrence of failures. Figure 5.13 shows the consumption of the system and the portion of power consumed because of

failures for different fault tolerance techniques. Since we found that many worker nodes were idle, we assumed to not add any machines specifically for the fault tolerance, but to exploit the idle ones. The consumption due to fault tolerance and the total consumption due to failures only include the dynamic power consumption of already present nodes implementing the fault tolerance, and not also the static consumption of additional nodes dedicated to the fault tolerance mechanism.

In the case of job **checkpointing**, consumption due to failures is reduced down to $1,577$ W, while the total consumption increases up to $42,579$ W (Figure 5.13a). The cost due to the machines adopted to implement the checkpointing strategy is about 157 W. As a consequence, most of the consumption is due to those failures not covered by the mechanism.

When adopting job **replication**, the cost of failures and fault tolerance strategy is $2,820$ W, and the total consumption is $43,353$ W (Figure 5.13b). In this case, most of the malfunction consumption is due to the replication rather than to not avoided failures ($2,400$ W and 420 W, respectively). The high cost of this technique is due to the increment of the number of running jobs of a fraction equals to the chosen coverage. Being in our case $cov = 0.5$, the running workload is 1.5 times the submitted one, sensibly incrementing the consumption of the system. It is worth to remark that we are assuming free nodes of the system being used to implement the fault tolerance. In Chapter 6, it is shown that, if using additional nodes, the difference of the impact of the replication on the total system consumption with respect to the checkpointing is even larger.

When using only **NOMAD**, the total consumption of the system presents a little growth up to $42,326$ W, while the consumption due to failures and to the tool itself is 841 W (Figure

5.13c). Unlike replication, in this case, most of this consumption is due to failed jobs, since the expected cost of NOMAD is just 73 W. Therefore, most of the total consumption due to the malfunction is caused by uncovered failures more than to the fault tolerance mechanism.

Finally, we evaluated the use of both job **checkpointing and NOMAD**. This case is similar to the one of using just checkpointing, but with different repair rates for queue and running failures (see Section 5.2.4). The consumption due to system malfunction is considerably reduced down to 334 W and the system consumption is $42,536$ W (Figure 5.13d). The power consumption due to uncovered failures is 168 W. This means that the effect of failures is largely reduced.

5.4.3 Availability

The availability of the system significantly changes in the various configurations. The **ideal system** is obviously 100% available (no job fails). Conversely, the **real system** is the one with the lowest availability (0.99221).

Checkpointing is not able to largely increase the availability of the system, which is only improved by 0.3% with respect to the real system without fault tolerance (from 0.99221 to 0.99541 - the difference is statistically significant at 95% level). This means that, in a system running 100,000 jobs per year, when using checkpointing, the expected number of jobs experiencing a failure every year is 459, while without fault tolerance it is 779.

Replication improves the availability of the system up to 0.99927. A pure performance

analysis would suggest to implement replication, since providing a significantly larger availability. The same conclusion would be reached with a performability analysis: as discussed in Section 5.4.1, the performance of the system is pretty the same in all the configurations, so the replication would be selected also in this case. Nevertheless, the consumption due to the mechanism is larger than the one due to the checkpointing, causing the increase of the total system consumption (the instantaneous power consumption goes from 42,279 W of the case without fault tolerance to 43,353 W). If this consumption increase can not be faced, adopting replication is not feasible. If, on the contrary, high availability is the focus, its implementation may help to reach the goal.

When using **NOMAD**, the availability of the system is 0.99637. This value is 0.1% better than the case of checkpointing. Also, both the consumption due to the mechanism and the total consumption due to failures are less than the case of job checkpointing. The total consumption of the system is reduced as well.

When using both **checkpointing and NOMAD** the expected steady state availability is 0.99985. In this case the total consumption due to failures is less than all the other cases. Thus, this solution appears to be more *consumable* than the others: *it provides a larger availability at a reduced cost, in terms of energy consumption, and does not affect the performance.* This is confirmed by the consumability metric η discussed in the successive Section 5.5.

5.5 Energy Efficiency and Consumability

As discussed in Section 1.3, energy efficiency of processing systems is usually computed as the relation between a performance metric and a consumption metric. Synthetically, energy efficiency e is given by:

$$e = \frac{P}{W} \tag{5.13}$$

We propose to evaluate the energy efficiency of processing systems, and the consumability as:

$$\eta = \frac{W - W_f}{W} \tag{5.14}$$

where W is the total power consumption of the system and W_f is the power consumption due to the malfunction of the system, which also includes the consumption for the fault tolerance technique, if any.

In this Section we show the limits of the common metric e and how the consumability metric η can help in analyzing the energy efficiency taking into account also the cost of the malfunction. e is computed as the ratio between the throughput of the system (expected number of jobs completed in an hour) and its expected total consumption.

Figure 5.14 compares the two metrics (black bars are related to e, white bars to η).

The first observation is that results achieved with e are very close to each other, even though we checked that these values are different at 95% confidence level (i.e, the confidence intervals for differences do not include 0). For instance, e is the same with and without

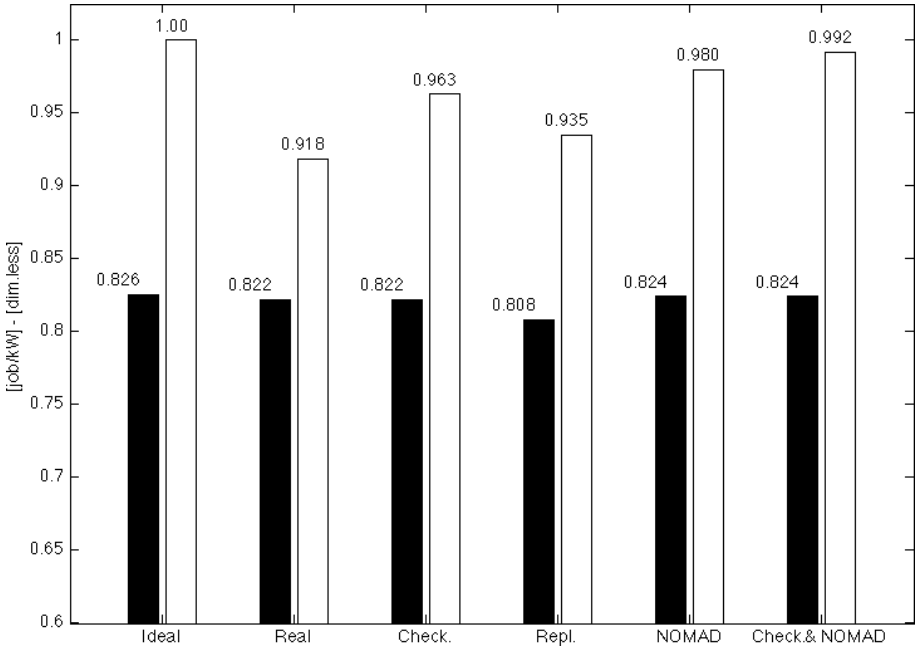


Figure 5.14: Values of the metrics e (black bars) and η (white bars) for various configurations.

checkpointing. Thus, it is not able to appreciate the reduction ($> 50\%$ in this case) of the electric power consumption due to failures. The best energy efficiency is obtained for the **ideal system**, when failures are disregarded ($e = 0.826$). When using only the **NOMAD** tool and when using both **NOMAD and checkpointing**, energy efficiency does not change. This is due to the small difference of power consumption and throughput achieved with these mechanisms. The large difference of the availability for the two configurations is not observable by means of e. The worst energy efficiency is corresponds to the **replication** ($e = 0.808$), which reduces the number of failures with respect to checkpointing and NOMAD alone (see Table 5.3). It is even less than the efficiency of the **real system** without any fault tolerance mechanisms. This is mostly due to the increase of the total consumption of about 1 kW for executing a larger number of jobs. Also in this case, the

performance-consumption ratio is not able to appreciate neither the reduction of the failure consumption nor the increment of the system availability up to three 9s.

These results show that a metric based only on performance and consumption does not allow us to measure the difference in exploiting system resources of various configurations. The administrator that has to select the management strategy will have only the chance to consider the ones not affecting performance but reducing energy consumption, or, vice versa, not affecting consumption and improving the performance. No quantitative information s/he will receive from e on how much the system is dependable, how much a failure costs, and how much fault tolerance costs. In this case study, replication appears to be completely not energy efficient, even less than the system without fault tolerance. This is strange if one observes that performance is unchanged, consumption increases by 2.5%, but availability grows up to 99.9%. Also, when using only the NOMAD tool or NOMAD and checkpointing together, the value of e is the same, since performance and consumption are similar in the two cases. On the contrary, the power consumption due to failures is reduced with the latter approach and the availability improves from 99.637% to 99.985%. In a system running 100,000 jobs per year, this corresponds to a reduction of the expected number of failed jobs from 363 to 15. If the goal of the administrator is a very highly available processing system to respect certain SLAs, s/he cares about availability trend and cost. In other terms, when evaluating a processing system, one can not disregard the failures.

To evaluate the impact of system malfunction on consumption and performance, and to

figure out the strategy that improves the efficiency of the system, we introduce η (Equation 5.14; see also Section 4.4).

The high correlation between the total consumption and the throughput of the system confirms that W is also representative of the performance of the system. This correlation is larger than 90% both with and without fault tolerance. The high (negative) correlation between the consumption due to failures and the availability confirms that W_f also contains information on how much the system is dependable. Also in this case, the correlation is larger than 90% both with and without fault tolerance.

The value of η for the different configurations of S.Co.P.E. is represented by the white bars in Figure 5.14. The maximum value is achieved for the **ideal system** ($\eta = 1.00$). Since no power is wasted because of failures, all the incoming power is transformed into useful work. Hidden costs due to failures are spotlighted by the reduced consumability of the **real system** without fault tolerance ($\eta = 0.918$). In fact, differently from Equation 5.13, η is sensible to failure consumption reducing the efficiency of the system. It also allows us to observe the improvements provided by the adoption of fault tolerance strategies.

An increment of η is observable when adopting **checkpointing** ($\eta = 0.963$). With respect to the system without fault tolerance, the total consumption increases by 300 W, but the power consumption due to failures is reduced by $1,870$ W. **Replication** appears to be less *consumable* than checkpointing ($\eta = 0.935$); however, the corresponding η is larger than the one of the system without fault tolerance (while the e was less). The low efficiency of this strategy is due to the cost of its implementation ($2,400$ W). Nevertheless, the consumption due to the malfunction (both failures and fault tolerance) is less than the cost due

to failures in the real system (2,820 W vs 3,447 W). Although not able to largely improve the dependability of the system, the simple **NOMAD** tool presents a high consumability ($\eta = 0.980$). This is mainly due to the reduced cost of the implementation of the technique (just 73 W), for which 98% of the absorbed electric power is exploited to complete jobs. The best consumability is measured when using **NOMAD and checkpointing** together. The low cost of the fault tolerance technique and the mitigation of the effects of a number of failures reduce the consumption due to malfunction, while the overall system consumption does not grow so much (as for replication).

The relation between the consumption due to failures and the overall system consumption allows one to estimate the ability of the system in exploiting available resources both to produce useful output and to mitigate failure effects. It overcomes the problem of constant system throughput by directly estimating the effect of the malfunction on the efficiency of the system. As a matter of fact, results show that the checkpointing mechanism and the NOMAD tool together represent the best choice in terms of fault tolerance to improve the availability of the system (hence to reduce the cost of the the failures) minimally affecting performance and total power consumption.

5.6 Discussion

The presented case study was useful to show the feasibility of the proposed solution to evaluate costs and benefits of several system management strategies considering performance,

energy consumption and dependability aspects simultaneously. Also, the obtained results highlighted the importance of the consumability analysis to figure out the real efficiency of a data center. The trend of the *performance* is not able to unearth neither undesired phenomena that take place in processing systems nor their cost. The same is for common *energy efficiency* metrics, which may even present misleading results. A *performability* analysis, while including both performance and availability aspects, neglects the energy consumption necessary for achieving such results. Hence, the evaluation of ***how much a system is consumable*** is peculiar to system administrators who are often expected to determine the soundness of decisions. They need a systematic process at this aim, which has been presented in Chapter 4 and exemplified in this Chapter.

In the case of S.Co.P.E. data center the results showed that replication is a good solution when large availability is the main objective of the management strategy and no cost restrictions exist. If energy consumption is a key factor of the analysis, and the aim is to make the system consumable, checkpointing and a good failure detector together are the best choice.

We stress that discussed results have been achieved by applying the consumability analysis described in Chapter 4 to a real system. Computed figures are specific to the S.Co.P.E. batch system, but it acts as a case study and validation means to demonstrate the importance of consumability evaluation and the applicability of the proposed analysis.

The detailed explanation of all the steps of the analysis helps to repeat it for other systems. The model easily fits all queue systems and simply requires identifying the kind of

jobs usually running in the system and their characterizing rates: arrival, scheduling, and completion. The presence of other kinds of jobs just requires complementing the overall model with an SRN with the appropriate rates and guard functions. The necessity for specializing the rates for each system is obvious and common to all models: each system has its own performance and usage features. The same is for the weights of the consumption model and for the failure rates. This also confirms the importance of the preliminary data analysis. The presentation in this chapter is meant to allow practitioners to easily replicate the proposed solution to other systems, thus fostering the quantification of the malfunction cost and the adoption of the proposed metric.

Finally, we emphasize the possible impact of the system dimension on the results. Since the probability of experiencing a failure grows with the system size, the larger the system, the greater the overall failure rate and maintenance costs. As a consequence, for a batch processing system bigger than the evaluated one (e.g., a system with thousands of worker nodes), but with similar hardware and configuration (common to modern batch systems), the behavior is likely to be very similar, and the percentage of the consumption due to failures to be the same or even larger.

Chapter 6

Case Study 2: Impact of Virtualization on Consumability

Virtualization is often adopted in processing systems to better exploit existing resources and to simplify the implementation of fault tolerance techniques. Cloud computing software platforms are then used to support the management of virtual machines, such as their placement, creation and disruption, and to configure the fault tolerance. In this chapter, we perform the consumability analysis of a virtualized scientific data center to estimate the impact of virtualization on its consumability. S.Co.P.E. data center is considered and the consumability model is lightly modified to take into account the steps of the cloud service life cycle, consumption induced by virtualization, and failures that can happen because of the added software layers.

6.1 Virtualized Processing Systems

Virtualization is often used in large data centers for several reasons. In 70s, IBM introduced it for provisioning resources of a single mainframe to several users [105]. Nowadays, it is exploited for the elasticity it provides in creating *ah hoc* configured machines that can be moved in the data center to satisfy some constraints [66, 28]. In Section 1.2.3, we discussed on placement and packing of virtual machines (VMs) onto physical machines (PMs) for improving the performance-consumption trade-off. Virtualization also eases the

implementation of fault tolerance techniques [96]. As an instance, in the case of message-passing applications, without virtualization, the implementation of checkpointing is very hard.

To manage VMs creation, migration, access and disruption, cloud computing platforms are commonly adopted. They provide a set of software components for the interaction with human operators (a web front end) or other systems (a set of APIs) for accepting requests, a scheduler, a hypervisor and, possibly, other management tools.

One of the main advantage commonly exploited in the case of virtualized processing systems, is to keep switched on only those servers actually running a VM. In other terms, running VMs are used for executing the current load of the system, PMs hosting VMs are switched on, the remaining ones are in stand-by and can be awoken if other machines are necessary to host additional VMs.

One of the drawbacks in the use of virtualization is the performance degradation it introduces. Since the hypervisor has to manage multiple requests for the underlying hardware that, abstracted, is provided separately to multiple VMs, an additional overhead is present. Furthermore, before operations can be performed, a VM is to be created. In the case of a batch processing system, this results in an additional time before the execution of a job can start.

The use of a cloud platform for managing VMs also implies failures that can manifest in the additional software layers. The fault tolerance, whose implementation can be eased, as always has a cost, in terms of consumed energy, which can be very large and thwart the benefits of the green management.

For the above mentioned issues, before adopting virtualization, a system administrator should properly evaluate its impact on the consumability of the system. As an instance, VMs can be differently sized (in terms of CPU and memory) or several scheduling of VMs on the servers may be adopted. Hence, the effect of those configurations on performance, consumption and dependability attributes should be assessed. Also, if fault tolerance is implemented, its tuning is paramount to make the system consumable by improving availability without affecting performance and consumption.

We performed the consumability analysis of a virtualized data center for processing batch jobs as discussed in Chapter 4. The target system is the real S.Co.P.E. scientific data center introduced in Chapter 5. For the model, we used some of the distributions estimated for S.Co.P.E. However, some changes were done to cope with the use of virtualization and of a cloud software platform.

6.2 Analyzing a Virtualized Batch System

We define the consumability model of the virtualized system by adapting the performance, consumption, and failure models presented in Section 4.2.

Before executing a job in a VM, that machine has to be instantiated on a physical machine. This operation is called *provisioning* of a VM. After the execution, the machine is to be deleted and resources of the PM freed. This is called *deprovisioning* of a VM [47]. The

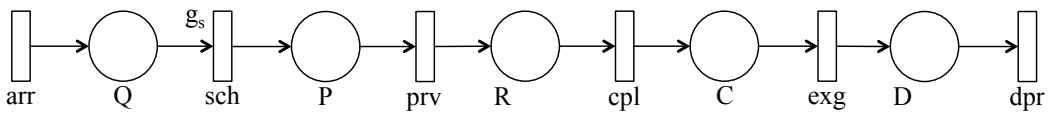


Figure 6.1: Performance model of a virtualized batch processing system.

performance model discussed in Section 4.2.1 is adapted to the virtualized system case by adding two transitions modeling the provisioning phase and the deprovisioning phase. This is depicted in Figure 6.1. Additional place-transition couples $P - prv$ and $D - dpr$ model the provisioning and deprovisioning, respectively. We assumed switched on only those PMs where at least one VM is running; the others are in stand-by, that is, they have a very reduced power consumption but require a certain time before a VM can be hosted. This requires to evaluate the distributions for the provisioning operation both on a switched on PM and on a stand by PM.

As for the consolidation (i.e. the packing of VMs on a small number of servers for reducing the overall power consumption), we considered the simple and widely adopted First Fit Decreasing (FFD) bin packing heuristic [137]. According to FFD, the next VM to be provisioned is always packed on the first server (the "bin") it fits; only when a server is filled up to its capacity, another server is used for scheduling.

For the fault tolerance, we did the same assumptions discussed in Section 5.2.4.

6.2.1 Performance Analysis

For the probability distributions of arrival and scheduling transitions, we adopted the same computed in the case of the non-virtualized S.Co.P.E. system (see Section 5.2.1), whereas

provisioning, deprovisioning and completion distributions were computed since specific to the virtualized case. We performed experiments on a S.Co.P.E. cloud prototype offering an Infrastructure as a Service (IaaS) (used for delivering processing capabilities) based on KVM[1] as hypervisor and OpenStack[2] as cloud software platform. Both of them are free and open source. We used different configurations of VMs:

- "*xlarge*": with 8 cores and 32 GB of memory;
- "*large*": with 4 cores and 16 GB of memory;
- "*medium*": with 2 cores and 8 GB of memory;
- "*small*": with 1 core and 4 GB of memory.

Provisioning and deprovisioning times

To evaluate the time required for the provisioning of a VM, we performed several experiments with a full factorial design with replication with factors (i) the **size of the VM** and (ii) the **status of the PM** where the VM is to be created. That is, we evaluated the creation of "xlarge", "large", "medium" and "small" VMs in a switched on and running PM or in a stand-by PM. Each experiment was performed 10 times. We estimated ì that 16.5 seconds were taken for the provisioning of a "small" or "medium" VM on a switched on and running PM, and 17.4 seconds were used for a "large" or "xlarge" VM. In the case of a stand-by PM, 107.9 seconds were taken for instantiating "small" or "medium"

[1] http://www.linux-kvm.org
[2] http://www.openstack.org

Table 6.1: Provisioning and deprovisioning distributions.

PM	**VM**	**Transition**	**Distribution**
Switched on	small, medium	*prv*	Exponential(218.18)
	large, xlarge		Exponential(206.90)
Stand-by	small, medium		Exponential(33.364)
	large, xlarge		Exponential(32.757)
-	-	*dpr*	Exponential(302.52)

VMs and 109.9 for "large" or "xlarge" VMs. To estimate the statistical significance of the results, we performed ANOVA analysis [69] and F-test (see also Section 5.2.2). For the size of the VM, we found F-computed=16.3>F-table=6.8; as for the status of the PM, F-computed=2165072>F-table=6.8.

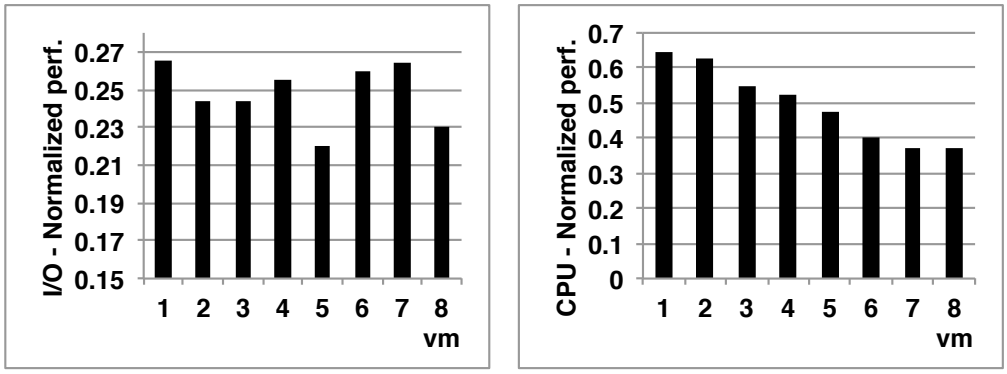
For modeling the different provisioning rates, the provisioning transition is make marking dependent with respect to the R place. Namely, if the modulus of the division of the number of running jobs by the number of cores per node (8) is different from zero, a new machine is to be switched on; in such a case, the provisioning time is larger.

We performed similar experiments for evaluating the deprovisioning time. We found that no factor affect it significantly more than the others. The mean time for the deprovisioning of a VM is 11.9 seconds.

The distributions used in the model are reported in Table 6.1 (parameters in hour^{-1}).

Evaluation of completion rates

For the job completion rates, we evaluated how the performance changes with respect to different configurations of number of VMs per PM, VMs sizes, and workload mix. All performance variations reported in this section are normalized to the performance of the PM



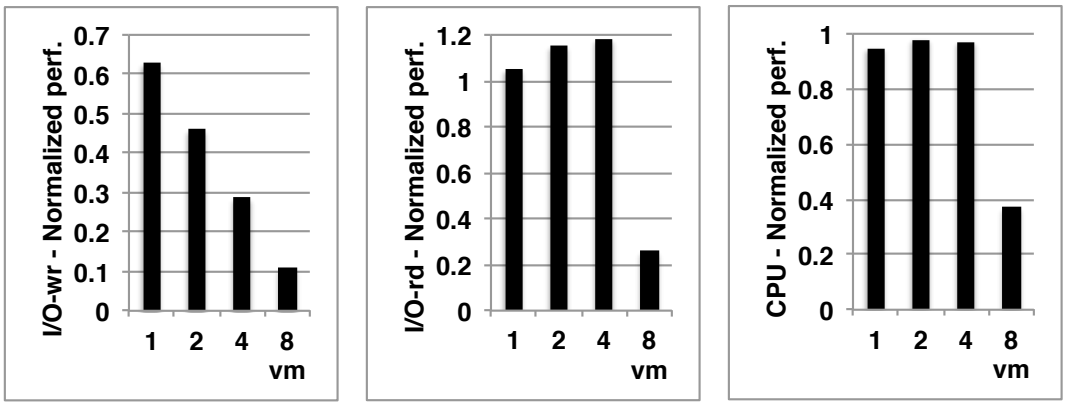
(a) (b)

Figure 6.2: Performance when varying the number of small VMs.

executing the same type and number of jobs.

Number of VMs. Figure 6.2 shows the performance variation as the number of small VMs running in the same PM increases. Each VM executes one job; values are normalized to the execution of the same number of jobs in a non-virtualized machine. Specifically, we considered from 1 to 8 VMs each running one I/O job (Figure 6.2a) or one CPU job (Figure 6.2b). In both cases, there is a significant performance loss. For I/O jobs, we do not have a clear trend, but the performance is reduced to about 25% of the execution on a PM, irrespective of the number of VMs. The situation does not change if assessing performance for I/O-read and I/O-write operations separately. For CPU jobs, performance decreases from 64% to 37% with respect to the non-virtualized system.

Size of VMs. When varying the size of VMs and the number of jobs running in each VM, the trend of the performance variation is very different. Figure 6.3 depicts the performance

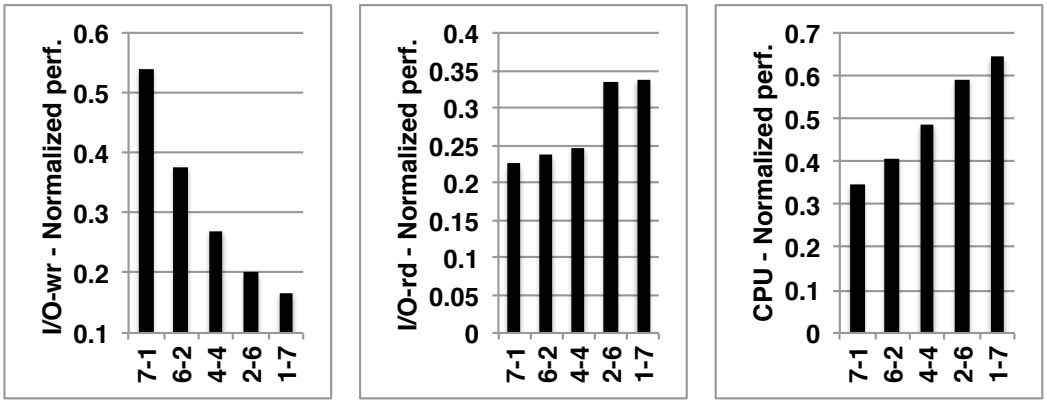


(a) (b) (c)

Figure 6.3: Performance when varying the size of VMs.

trend as the number of VMs varies from 1 xlarge to 8 small, each executing 8, 4, 2, 1 jobs, respectively, normalized to the execution of 8 jobs in a non-virtualized machine for several types of jobs: (a) I/O jobs performing write operations, (b) I/O jobs performing read operations, and (c) CPU jobs. While I/O-write jobs present a decreasing performance trend (Figure 6.3a), I/O-read jobs may perform even better on the virtualized system than on the PM, depending on the configuration (Figure 6.3b). In particular, up to 4 medium VMs executing 2 jobs each, the virtualization allows the system to increase the parallelization of read operations, improving the overall performance. In the case of 8 small VMs each running 1 job, the performance decrease is evident, instead. A similar trend is observable for CPU jobs (Figure 6.3c).

If not differently specified, by I/O load we mean a load composed by 80% of read operations and by 20% of write operations, as in the case of S.Co.P.E. processing system.



(a) (b) (c)

Figure 6.4: Performance when mixing 8 small VMs with different load.

Workload mix. We also observed that workload mixes, i.e. the scheduling of VMs executing different types of jobs (e.g., CPU bound jobs or I/O bound jobs) on the same PM, can be beneficial in terms of performance.

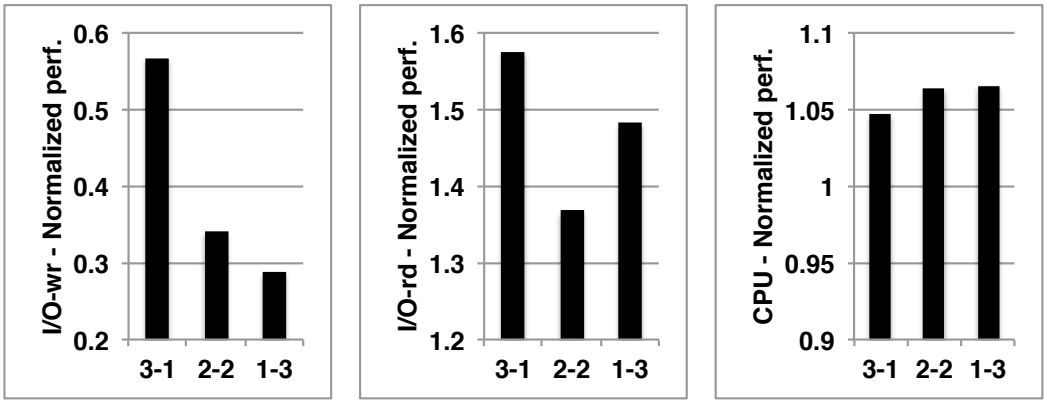
First, we evaluated the mix of **8 small VMs** on the same PM. Figure 6.4 reports the performance for various job mix in 8 small VMs running on the same PM, normalized to the execution of 8 jobs in a non-virtualized machine for different types of jobs: (a) I/O jobs performing write operations, (b) I/O jobs performing read operations, and (c) CPU jobs. x-y indicates the number of machines executing CPU jobs and I/O jobs, respectively.

Figure 6.4a shows the performance variation of I/O-write operations when considering from 7 VMs executing 1 CPU job each and 1 VM executing 1 I/O job, to 1 VM executing a CPU job and 7 VMs executing 1 I/O job each. The greater the number of VMs executing I/O jobs, the worse the write performance. On the contrary, as shown in Figure 6.4b, when decreasing the number of machines executing CPU bound jobs, read operations perform better. This suggests that the use of the CPU significantly affects read operations, while

write operations are mutually affected each other (also for these results, statistical significance was tested through the F-test). In fact, also in Figure 6.3b, we observed how the virtualization is able to increase the parallelism of read operations up to improve the performance with respect to bare metal machines. Figure 6.4c shows that also the performance of CPU bound jobs improves when increasing the number of I/O jobs against CPU jobs. It is worth noting that in all such cases the performance get much worse with respect to the execution of jobs on a PM or on a reduced number of VMs (see Figure 6.3).

We studied the effect on performance of various job mix in **4 medium VMs** running on the same PM, normalized to the execution of 8 jobs in a non-virtualized machine. Results are reported in Figure 6.5 with respect to (a) I/O jobs performing write operations, (b) I/O jobs performing read operations, and (c) CPU jobs. x-y indicates the number of machines executing CPU jobs and I/O jobs, respectively. Each VM executes 2 jobs of the same kind. Specifically, we considered from 3 VMs executing 2 CPU jobs each and 1 VM executing 2 I/O jobs to 1 VM executing 2 CPU jobs and 3 VMs executing 2 I/O jobs each. Interestingly, read operations and CPU bound operations get better on VMs than on a PM, thanks to the parallelization of read operations in the VMs and their interleaving with CPU operations.

The performance for I/O-write operations decreases as the number of machines executing I/O jobs increases, as depicted in Figure 6.5a. Nevertheless, read operations and CPU bound operations get better than when executing in a PM. Figure 6.5b shows that performance of I/O-read bound jobs are increased by at least 34% when running 4 VMs for each PM and mixing running jobs. Performance of CPU jobs improves by about 6%, as shown in Figure 6.5c.



(a) (b) (c)

Figure 6.5: Performance when mixing 4 medium VMs with different load.

These results highlight the importance of properly tuning virtualization in terms of size of VMs and of the scheduling based on the possible workload mix in order to avoid performance degradation, while exploiting the management elasticity. The impact of the VM size and of the scheduling on the consumability is discussed in Section 6.3.

6.2.2 Consumption Analysis

The virtualization impacts on the consumption of the server hosting VMs. Hence, the weights of the model in Equation 5.9 were modified to account for the effects of virtualization and server consolidation. Furthermore, we assumed *switched on* only those servers executing VMs, while the rest were assumed to be in *stand-by* (i.e., with a reduced power consumption, but requiring a time before being operational and ready to host VMs).

We assumed an increase of the dynamic part by 7% for each VM [141], while for stand-by PM we considered only a static consumption of 7 W [64].

6.2.3 Failure Analysis

Concerning the failures that can happen in a virtualized environment, we did some studies to figure out which of the ones discussed for non-virtualized batch systems are likely to happen in a virtualized one, and what additional failures can happen in a virtualized system.

Failures specific to non-virtualized systems

We noted that the permanently or temporarily blocking of jobs, i.e. queue and running failures, are typically due to the gradual error accumulation within basic software of the processing system causing performance degradation trends (see Section 4.2.3). The problem lasts until the reboot of the nodes, which interrupts the performance degradation trend. Since VMs are booted every time a new job is started, we assumed these failures being absent in the model of a virtualized system.

Also misconfigurations, typically due to wrong settings made on a machine during its use, are removed by instantiating always the same VM image, hopefully correctly configured. Thus, we assumed that exiting failures do not take place in a virtualized environment.

Similar observations on failures and availability trends in virtualized environments are surveyed in [107].

Failures specific to virtualized systems

To find out during which steps of the job life cycle in a virtualized environment failures are likely to manifest, we also performed robustness testing of a cloud computing platform. Specifically, we forced both the input (e.g., type of machine to be created) and the state (e.g. number of VMs that are being created at the same time, outcome of a VM image creation, number of virtual machines concurrently running on a physical machine) of the system to figure out when failures can happen.

The typical approach for **robustness testing** aims at breaking the system by injecting unexpected and invalid inputs at its interface. Tests are generated by combining, with some strategy, such *exceptional inputs*, and then submit each test case rigorously in a clean initial state, by restarting the system each time. On one hand, this avoids a test case outcome to depend on the history of testing, thus easing the debug. On the other hand, treating the system as a *stateless* black box constitutes also the theoretical and practical limit of current robustness testing. Resubmitting the same inputs under different states is likely to expose new and more subtle failures. This is especially true for cloud computing systems, where phase-based interactions between multiple concurrent users and the platform entail strong dependencies of a request outcome on the state.

Specifically, a cloud system concurrently provides services to a set of users $U = \{u_1, u_2, \ldots, u_l\}$, each one submitting (a subset of) m possible request types, $R = \{r_1, r_2, \ldots, r_m\}$. We name a complete interaction of each user with the platform as *session*. A session (s_k)

evolves through a set of n *phases* $P = \{p_1, p_2, \ldots, p_n\}$. The state of a session at time t is thus characterized by the phase ($p_{ik}(t)$, $i \in \{1, \ldots, n\}$ - the k-th session is in phase i at time t) and by the request type being served ($r_{jk}(t)$, $j \in \{1, \ldots, m\}$ - the k-th session is serving the request j at time t). The **state** of the platform at time t is given by the union of the states of the active sessions: $\cup\{s_k(t)\} = \cup\{p_{ik}(t), r_{jk}(t)\}$. The number of active sessions implies the concurrency degree at a given time. The whole set of states is therefore given by the number of possible request types, multiplied by the maximum number of active sessions, and by the number of phases.

In the case of the virtualized batch processing systems, the final user is another system requesting for the creation of VMs which jobs will be scheduled on and executed in. Apart from the provisioning of the VM, other types of requests are the access to an operational VM and the deprovisioning of a VM. Also, the number of phases is reduced with respect to the case of providing a pure cloud service. As an instance, there is no authentication or VM reservations management.

We applied the state-based robustness testing to Apache Virtual Computing Lab (VCL), an open source cloud platform that allows users to create a wide range of solutions in terms of *IaaS*, *PaaS*, and *SaaS* [4]. We created a VCL testbed in our laboratories, with two machines, acting as controller node and hosting node, interconnected by a LAN, equipped with: Apache VCL 2.3, VMware server 2 as hypervisor, CentOS 6 as OS on the two machines and both Linux and Windows images for guest machines.

62 failures were exposed when testing the ***AddHost*** functionality. It consists in adding to the system a new node where VMs can be created and is related to the management phase, which includes operations such as upgrades, backup, failure/repair/migration of components. 40 of those failures appeared to not be dependent on the state, but only on the input; the state-dependent failures were mainly related to concurrency issues. However, in a virtualized batch processing system, we assume the *AddHost* operation to be performed once and not concurrently.

Testing of the ***ResourceRequest*** functionality exposed 24 failures. In this case, all the exposed failures were only input-dependent. That is, none of the exposed failures was due to concurrency or other particular states of the cloud platform. It is reasonable to expect that in the virtualized batch system scenario, the request for a virtual resource is always performed in the same way, with the predefined input parameters representing the fixed characteristics of VMs.

We also found that the creation of a VM, performed by the ***CreateVM*** during the provisioning phase, often fails and in about 50% of the cases (23 over 47) the failure was state-dependent. As an instance, the provisioning of a VM may fail if many VMs are created at the same time, if management operations (e.g., adding a new host or removing an existing host) are performed concurrently with the VM creation, or if the VM is scheduled on a host not properly configured. The VM creation is performed continuously in a virtualized batch system, to host the execution of submitted jobs. Then, the **provisioning failure** is certainly to be taken into account.

For the ***DeleteVM*** functionality, related to the deprovisioning phase, the testing exposed failure 19 times and all were state-dependent (the function has as unique input the identifier of the VM to be destroyed, which is automatically provided by the front end, thus, input-dependent defects are very difficult to be found in this case). For example, system resources may be not freed after the request for deleting a VM. Also this operation is performed continuously in a virtualized batch system, therefore also the **deprovisioning failure** is to be considered.

Unfortunately, we did not have data from a real environment to estimate the actual rates for provisioning and deprovisioning failures. We assumed the same distributions of queue and exiting failures of the non-virtualized system for provisioning and deprovisioning failures, respectively, so that the order of magnitude of failure rates is the same for the two systems and results are comparable.

The **abort** of running jobs can happen in a virtualized environment as in a non-virtualized one. However, in the virtualized system, we excluded those causes in common with queue failures. Therefore, the abort failure rate is $\lambda_{a_{logs}} = 0.00318$ (hour^{-1}) corresponding to the abort rate as found in the system and scheduler logs (see Section 5.2.3).

Figure 6.6 depicts the performance and failure model of a virtualized batch system. Transition prv and place P model the provisioning, dpr and D model the deprovisioning. Places PF, A, and DF model the provisioning, abort, and deprovisioning failures,

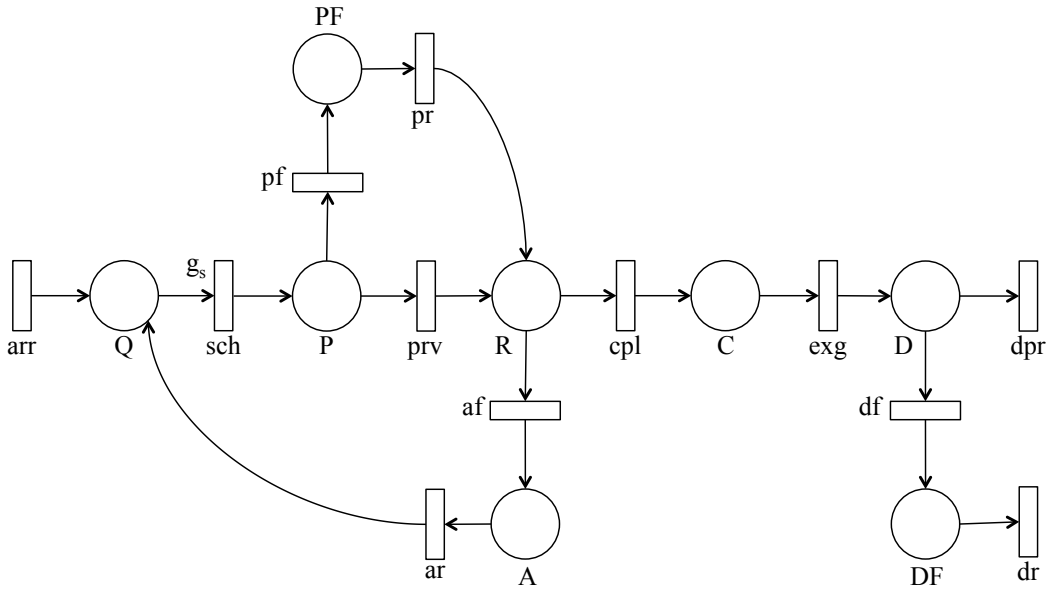


Figure 6.6: Performance and failure model of the virtualized system.

respectively. While a token is in the P place waiting for provisioning transition to fire, the provisioning failure transition pf may fire; this removes a token from place P and adds a token to place PF, modeling the failure of the provisioning phase. The repair is modeled by removing a token from place PF and moving it to place R. The abort is modeled by moving a token from place R to A, upon firing of transition af. The repair requires the resubmission of the request; this is modeled by transition ar, which, upon firing, moves a token from A to Q. Finally, the deprovisioning is modeled by transition df that removes a token from place D and adds a token to place DF, upon firing. Transition dr models the repair by simply removing a token from place DF.

Although the model validated in the previous Section 5.3 is lightly modified, we are confident that by adding two transitions, the SRN is still able to approximate the average

behavior of the real system if applying virtualization, in terms of performance, power consumption, and availability [128].

The parameters of the model specific to the virtualized batch processing system are summarized in Table 6.2. Distribution parameters are in seconds for arrival, scheduling, completion and exiting distributions and in hour^{-1} for the failures (for better readability). Consumption is in Watts. The coefficients for the performance model distributions are the ones estimated by means of the performance analysis discussed in Section 6.2.1; for the performance, HPC jobs are modeled in the same way as CPU jobs. The weights for the consumption model are computed incrementing by 7% the dynamic consumption as discussed in Section 6.2.2.

6.2.4 Fault Tolerance

For the fault tolerance, we evaluated job checkpointing and job replication as in the case of the non-virtualized system. They are also modeled as in the case of non-virtualized system and same assumptions were done for checkpointing time, network latency, and consumption (see Section 5.2.4). As for checkpointing frequency, and checkpointing and replication coverage, here several values are considered, as detailed in the following Section 6.3.

Table 6.2: Transition distributions and weights of the virtualized batch processing system model.

Job type	**Transition**	**Distribution**
IO	arr_{IO}	LogNormal(2.4981, 2.2846)
	sch_{IO}	Exponential(48.196)
	cpl_{IO}	LogNormal(6.4649, 1.4964)
	exg_{IO}	Exponential(30.0)
	cpl_{IOvm} - with 8 VMs	$cpl_{IO} \cdot 0.23$
	cpl_{IOvm} - with 4 VMs	$cpl_{IO} \cdot 1.00$
	cpl_{IOvm} - with 4 VMs (mix 3-1)	$cpl_{IO} \cdot 1.38$
	cpl_{IOvm} - with 4 VMs (mix 2-2)	$cpl_{IO} \cdot 1.16$
	cpl_{IOvm} - with 4 VMs (mix 1-3)	$cpl_{IO} \cdot 1.24$
CPU	arr_{CPU}	LogNormal(4.1366, 1.9643)
	sch_{CPU}	Exponential(214.87)
	cpl_{CPU}	Gamma(0.40369, 54486, 0)
	exg_{CPU}	Exponential(30.0)
	cpl_{CPUvm} - with 8 VMs	$cpl_{CPU} \cdot 0.37$
	cpl_{CPUvm} - with 4 VMs	$cpl_{CPU} \cdot 0.97$
	cpl_{CPUvm} - with 4 VMs (mix 3-1)	$cpl_{CPU} \cdot 1.05$
	cpl_{CPUvm} - with 4 VMs (mix 2-2)	$cpl_{CPU} \cdot 1.06$
	cpl_{CPUvm} - with 4 VMs (mix 1-3)	$cpl_{CPU} \cdot 1.07$
HPC	arr_{HPC}	LogNormal(2.6263, 2.9161)
	sch_{HPC}	Exponential(8538.4)
	cpl_{HPC}	LogNormal(7.7713, 2.0656)
	exg_{HPC}	Exponential(30.0)
	cpl_{HPCvm} - with 8 VMs	$cpl_{HPC} \cdot 0.37$
	cpl_{HPCvm} - with 4 VMs	$cpl_{HPC} \cdot 0.97$
	cpl_{HPCvm} - with 4 VMs (mix 3-1)	$cpl_{HPC} \cdot 1.05$
	cpl_{HPCvm} - with 4 VMs (mix 2-2)	$cpl_{HPC} \cdot 1.06$
	cpl_{HPCvm} - with 4 VMs (mix 1-3)	$cpl_{HPC} \cdot 1.07$
-	pf	Exponential(0.00130)
	pr	Exponential(0.01096)
-	af	Exponential(0.00318)
	ar	Exponential(0.1)
-	df	Exponential(0.01950)
	dr	Exponential(0.1)
	Weight	**Value**
-	W_s	156.91
IO	W_{IO}	4.09
	W_{IOvm} - with 8 VMs	6.38
	W_{IO} - with 4 VMs	5.24
CPU	W_{CPU}	10.47
	W_{CPUvm} - with 8 VMs	16.33
	W_{CPU} - with 4 VMs	13.40
HPC	W_{HPC}	13.58
	W_{HPCvm} - with 8 VMs	21.18
	W_{HPCvm} - with 4 VMs	17.38

Table 6.3: Design of Experiments: factors and respective levels.

Factor	**Level 1**	**Level 2**	**Level 3**	**Level 4**	**Level 5**
Virtualization	NO	YES	–	–	–
VM size	small	medium	large	xlarge	–
Scheduling	3-1 (4 VMs)	2-2 (4 VMs)	1-3 (4 VMs)	–	–
Fault tolerance	NO	Check.	Repl.	–	–
Checkpointing *cov*	0.5	0.8	1.0	–	–
Checkpointing *int* **[m]**	5	30	60	120	–
Replication *cov*	0.2	0.4	0.6	0.8	1.0

6.3 Experimental Results

We performed experiments for evaluating, at the steady state, the impact of the virtualization on the system behavior under several tunings and configurations. As in Chapter 5, we simulated the model with SPNP with the method of batch means, and computed results with 95% confidence intervals with maximum half-width of 5%. For all the results we checked the statistical significance by means of the F-test or the t-test. To check the significance of the difference between two results, the t-test for the mean difference was adopted (see Section 5.4).

Specifically, we used a factorial design with replication to study the response of the system in terms of mean job duration, total power consumption, power consumption due to failures (including the consumption of the fault tolerance mechanism, if any), and availability. Among all the factors, we discuss the primary ones, i.e. the factors that more significantly affect the response variables. These factors, along with respective levels, are summarized in Table 6.3.

The first factor simply represents if ***virtualization*** is adopted or not, and it is used to compare the trends of the virtualized system with the ones estimated for the non-virtualized

one. The ***VM size*** factor was used to estimate the impact of different virtualization configurations on the consumability aspects of a batch processing system. We evaluated the four configurations already discussed in Section 6.2.1. The factor ***scheduling*** allowed us to study the effect of a scheduling strategy running workloads of different kinds in a PM, isolating them in several VMs. We assessed the three cases with 4 medium VMs discussed in Section 6.2.1, since these provided promising results in terms of performance and number of VMs allowing us to study more mixtures. The ***fault tolerance*** factor represents the use of fault tolerance mechanisms in the virtualized system. It is useful to assess the system when no fault tolerance, checkpointing or replication is adopted. It is also interesting to evaluate the effect of several configurations of the adopted fault tolerance techniques. They can be tuned in terms of coverage and interval between successive checkpointing operations; hence, we included the factors ***checkpointing cov***erage, ***checkpointing int***erval, and ***replication cov***erage (respective level combinations can be also considered as levels of the *Fault Tolerance* factor, as for the levels of the *VM size* factor with respect to the *virtualization* factor; they are sketched in this way for improving readability).

6.3.1 Impact of VM Size

Figure 6.7 depicts the expected values of (a) mean job duration, (b) power consumption, (c) availability and (d) consumption due to failures for the non-virtualized system and for the virtualized one when adopting 1 x-large, 2 large, 4 medium, or 8 small VMs per PM.

Performance, depicted in Figure 6.7a as mean job duration, worsen when adopting

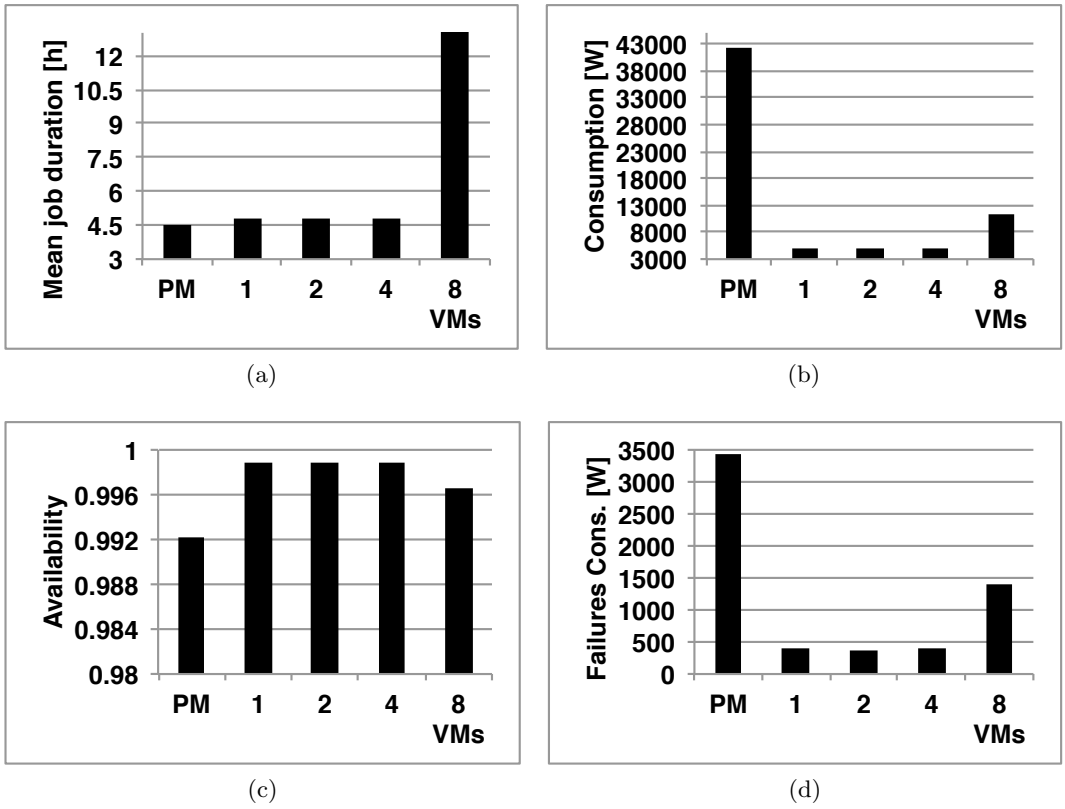


Figure 6.7: Impact of virtualization with VMs of different sizes.

virtualization as the number of VMs per PM increases. When running 8 small sized VMs on each PM, the expected job duration is about three times the one estimated for the bare metal machines. When running 1 x-large, 2 large or 4 medium PMs, performance is comparable to the case of the PM: for the PM, the mean job duration is 4.5123 h, while in the case of 1 xlarge VM executing 8 jobs it is 4.8392 h.

The considerable reduction of the power **consumption** leaps out in Figures 6.7b comparing the cases of the execution on (always switched on) PMs and on differently sized VMs. In the case with 8 small VMs per PM, the consumption is reduced by about 70%, from 42,279 W to 11,318 W. The management elasticity provided by virtualization permits the

packing (i.e., consolidation) of the load on a reduced number of servers, indeed. If assuming the other ones in stand-by, the electricity consumption is largely reduced with respect to the case of the non-virtualized system. While a similar management can be implemented for a non-virtualized system, cloud software platforms commonly provide means for switching on/off machines and reducing the energy consumption. On the other hand, the performance degradation heavily reflects on the whole system. With 8 VMs, the increase of the expected job duration makes the consumption larger than when using a reduced number of VMs per PM. If adopting just one xlarge VM, the consumption is reduced by more than 80% with respect to the non-virtualized system.

We also observe the improvement of the **availability**, depicted in Figure 6.7c. This is a consequence of the absence of gradual error accumulations and misconfigurations, as discussed in Section 6.2.3. Also, when using more small VMs, while increasing the elasticity, the reduced duration of the jobs (with respect to the case of 8 VMs) reduces the probability of aborts. As a matter of fact, the availability of the system with large or medium VMs is larger than the one of the system with small VMs. This also reflects on the expected **consumption due to failures**, shown in Figure 6.7d.

These results show that the virtualization eases the improvement of consumption and availability, but a wrong configuration may significantly deteriorate the performance. *The size of the implemented VMs is a key factor for the quality (in terms of performance and availability) and the efficiency (in terms of power consumption) of the processing system.*

We remark that, when choosing the size and number of VMs it is also important to

take into account the granularity that can be achieved with a large number of VMs. As a matter of fact, this provides more elasticity that may simplify the management of the workload, for instance providing more degrees of freedom in the implementation of consolidation strategies. On the contrary, just one large VM prohibits to execute different types of jobs in different and isolated VMs, or to migrate a set of jobs running in a VM from a PM to another one. In our settings, using 4 medium VMs seemed to provide a good trade-off between elasticity and consumability.

6.3.2 Impact of Scheduling Strategies

The results discussed in Section 6.2.1 showed that the right scheduling of VMs executing different kinds of operations can mitigate the performance degradation caused by virtualization or even improve the performance with respect to a PM executing all jobs of the same type.

To assess the impact of scheduling VMs executing different types of jobs on the same server, we considered the three configurations with 4 medium sized VMs. Figure 6.8 reports the results for the expected (a) mean job duration, (b) power consumption, (c) availability and (d) consumption due to failures; x-y indicates the number of machines executing CPU jobs and I/O jobs, respectively. Each VM executes 2 jobs of the same kind.

Performance improves in the 3-1 case, where 3 VMs execute 2 CPU jobs each and one executes 2 I/O jobs. In this case, the expected mean job duration is 4.3021 h, while it is the same for the 2-2 and 1-3 configurations (confidence intervals for the differences include

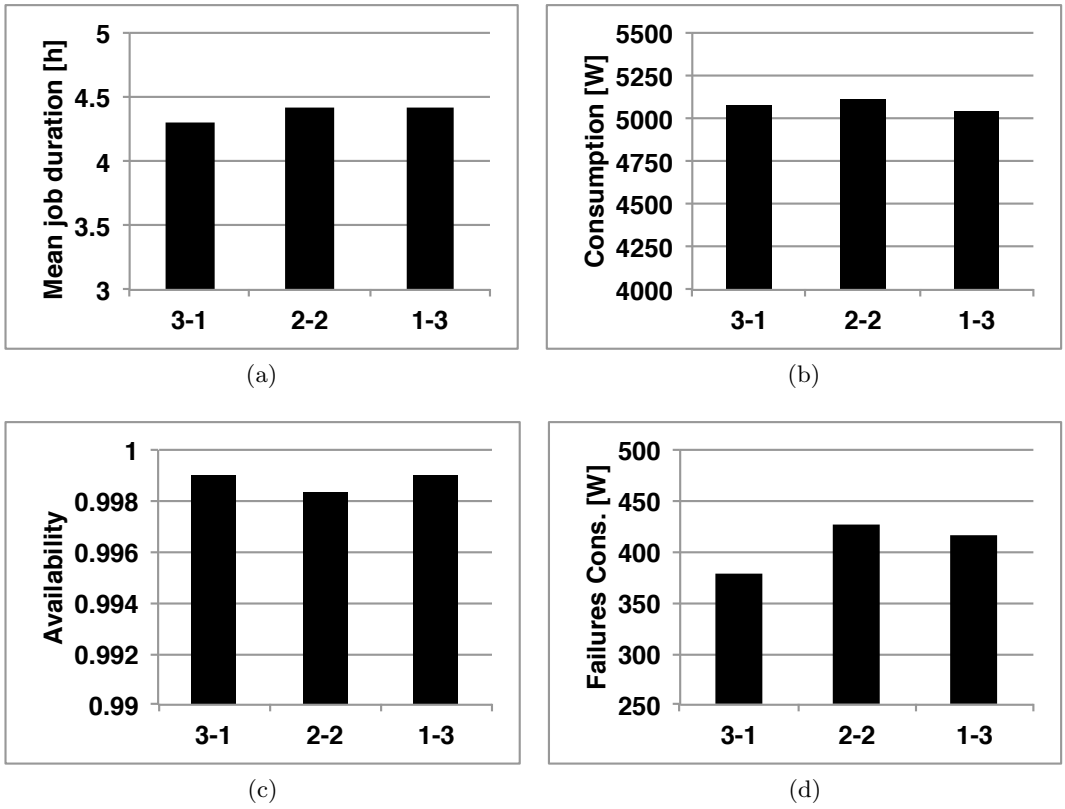


Figure 6.8: Impact of scheduling VMs executing different types of jobs on the same PM.

zero). With respect to the case of using a PM to execute only jobs of the same kind, the performance significantly improves up to 4.5%.

From the **consumption** point of view, the three configurations are the same (confidence intervals for the differences include zero). The **availability** improves with respect to the non-virtualized system. In the 3-1 case, the availability is 0.99902, while it was 0.99221 in the case of the non-virtualized system. As a consequence, also the **consumption due to failures** is reduced in the 3-1 case, which is 379.12 W, while it is 429.85 W in the 2-2 case.

This result is also reflected by the consumability η for the three mixtures of VMs, which is illustrated in Figure 6.9. The values of η for the mixtures 2-2 and 1-3 are the same

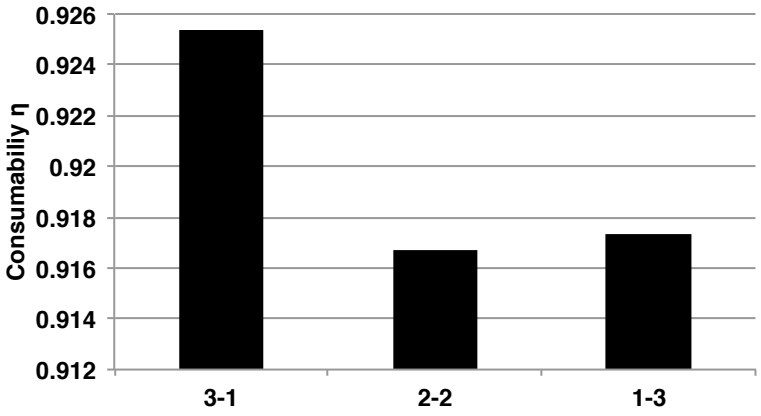


Figure 6.9: Consumability η when varying the scheduling of VMs.

(the confidence interval for the difference, given by the linear relation in η of the total consumption and the consumption due to failures, includes zero). The consumability of the 3-1 configuration is significantly better than the others at 95% confidence level.

The consumability analysis highlights how *properly scheduling VMs with different loads on the same PM may result in an improvement of both performance and availability, without affecting the power consumption.*

6.3.3 Impact of Fault Tolerance Strategies

We also assessed the impact of fault tolerance strategies to figure out their best tuning in terms of consumability. Similarly to what seen in for the non-virtualized system, we expect replication being able to provide a large availability improvement at the expense of a large consumption growth. Then, an administrator would look for a more consumable technique, which is able to find a good trade-off between availability and consumption, without affecting performance. While common analysis would only show that the performance remains

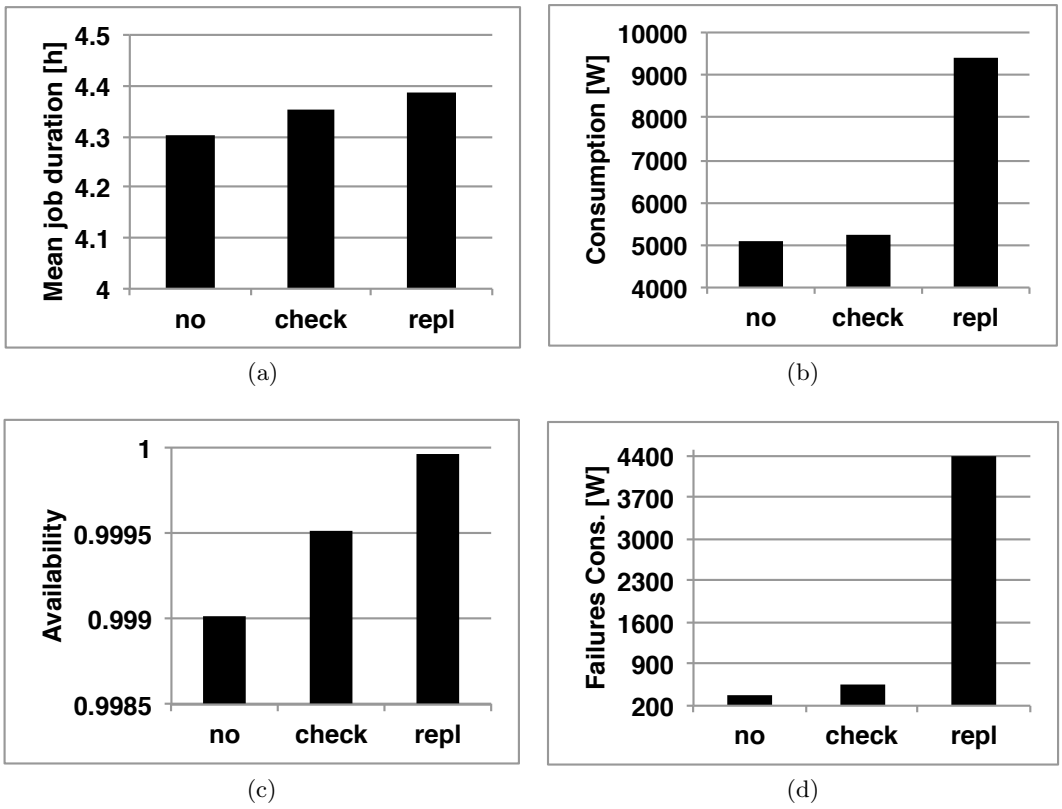


Figure 6.10: Impact of job checkpointing and job replication on the virtualized system.

unchanged and the availability improves, we also evaluate the impact on consumption and which portion of it is used by the fault tolerance: the one saved for the reduced number of failures or additional energy previously not required.

We considered the case of 4 medium VMs per PM, three of which executing 2 CPU jobs each, and one executing 2 I/O jobs. As discussed in Section 6.3.2, this configuration provided the best trade-off among performance, availability and power consumption. For these first experiments on fault tolerance, we assumed a coverage of 80% and a frequency of 1/30 minutes for checkpointing, and 80% coverage for replication. The results for the expected values of (a) mean job duration, (b) power consumption, (c) availability and (d)

consumption due to failures (including the consumption of the fault tolerance mechanism, if any) are depicted in Figure 6.10.

From the figure, we can observe that *the choice of the fault tolerance strategy strictly depends on the trade off between consumption and availability.* Performance is not different from one case to another at 95% confidence level (confidence intervals for the differences include zero). The availability estimated when adopting the replication is much greater than other cases, up to reach 0.99996. A pure availability analysis or a performability analysis would suggest to adopt replication. Nevertheless, the power consumption when replication is implemented is larger than the case with checkpointing (9,414.2 W vs. 5,243.6 W), because of the cost of the fault tolerance mechanism itself. In fact, also the consumption due to failures (which includes the power consumption caused by the machines used for the replication) is larger than the one estimated in case of checkpointing, despite the reduction of the number of failed jobs (availability is larger). Compared to the previous case study (see Section 5.4.2), here, the difference of the costs of the two fault tolerance techniques is even larger, given the additional nodes to be switched on for running replicated jobs. Checkpointing presents a small variation of the power consumption with respect to the case of no fault tolerance mechanism, and the availability is improved anyway. The consumability analysis points out the ability of the checkpointing to exploit the input power to produce useful output, i.e. to improve the consumability of the system.

Other experiments were performed for improving the consumability with different configurations of the fault tolerance.

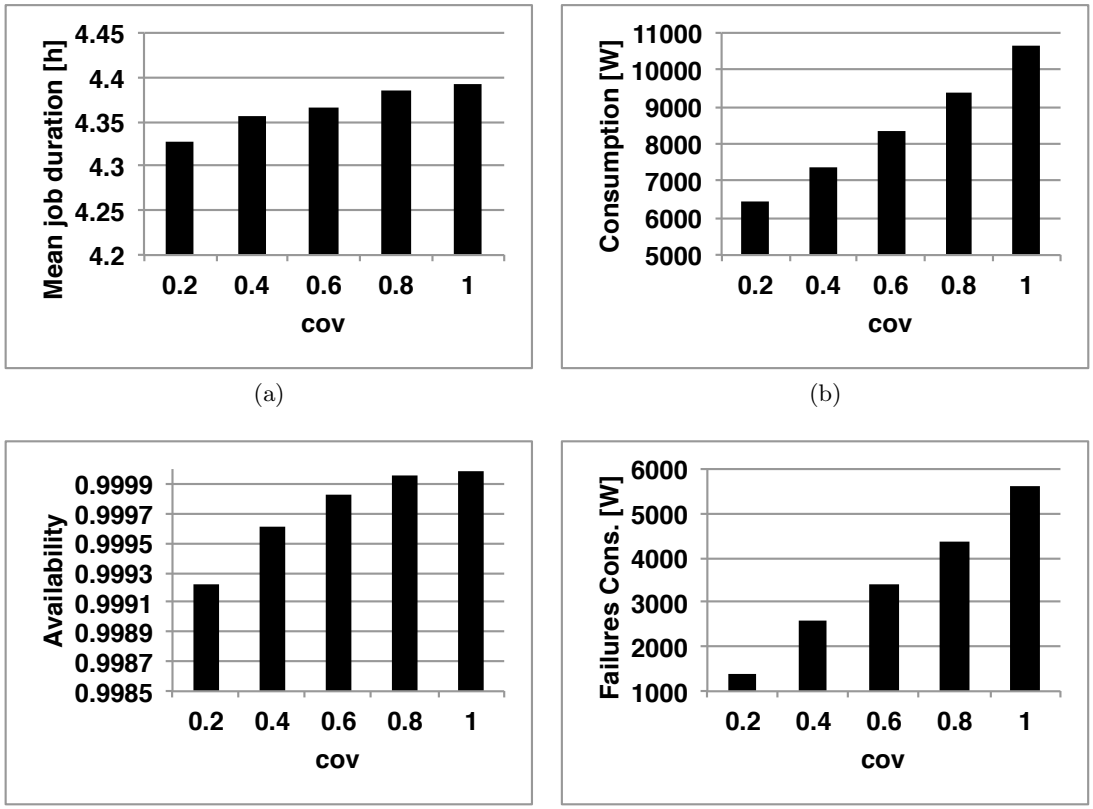


Figure 6.11: Results when varying the replication coverage.

Replication

In Figure 6.11, it is shown the effect of the replication mechanism for different values of the coverage on the expected values of (a) mean job duration, (b) power consumption, (c) availability and (d) consumption due to failures (including the consumption due to the fault tolerance mechanism). Performance remains in the order of 4.35 hours for all the cases. The consumption grows as the coverage increases, as depicted in Figure 6.11b. When $cov = 0.2$, the power consumption is $6,471.2$ W, which increases up to $10,638$ W when the coverage is 100%. This is due to the execution of the same jobs twice, as also

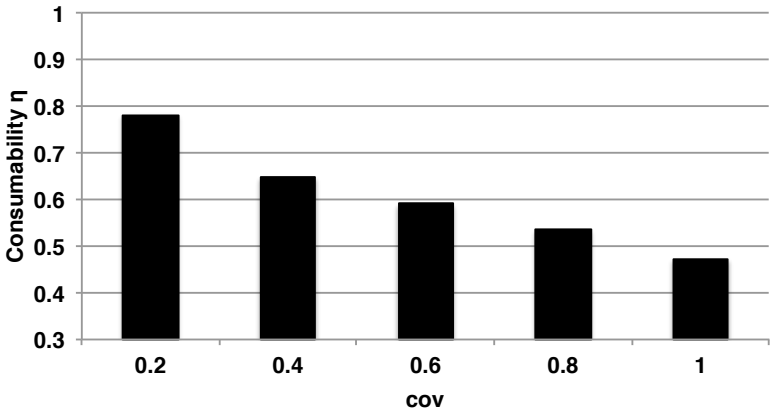


Figure 6.12: Consumability η when varying the replication coverage.

confirmed by the consumption due to failures, reported in Figure 6.11d. It reaches 5608.5 W with the maximum coverage and the consumption due to failures is larger than the total system consumption without any fault tolerance techniques (5,077.7 W). These results would suggest to always adopt a small coverage in order to reduce the power consumption. Clearly, also the availability, in Figure 6.11c, grows significantly from one configuration to another, up to 0.99999 in the case of $cov = 1.0$.

Those trends remark that replication is not consumable: whatever improvement in the availability of the system is obtained with the increase of the energy consumption. Hence, *job replication with high coverage is the fault tolerance strategy to adopt only if aiming at implementing a highly available processing system.* If reduced consumption is required, it can not be employed, since unable to exploit the energy saved for reducing the number of failures, requiring a large amount of energy for producing always the same quantity of useful work.

This is also evident from the trend of the consumability η (see Sections 4.4 and 5.5)

illustrated in Figure 6.12. It presents a decreasing trend with respect to the increment of the coverage. In the best case, $\eta = 78\%$; for $cov = 1.0$, η decreases down to 47%. That is, less than half of the incoming power is used for producing useful work; even though the consumption directly due to failures is less than 1%, more than 52% of the total power consumption is due to the fault tolerance.

Checkpointing

We estimated (a) mean job duration, (b) power consumption, (c) availability and (d) consumption due to failures (which includes the consumption due to the fault tolerance mechanism) for different values of checkpointing coverage and interval reported in Table 6.3. The results are shown in Figure 6.13.

The availability improves when the checkpoints are performed more frequently (i.e., the interval is reduced from 120 minutes to 5 minutes) and when the coverage increases from 0.5 up to 1.0, as shown in Figure 6.13c. In particular, availability goes from 0.99929 to 0.99989 (values are different at 95% confidence level). While this result is intuitive, the consumability analysis also highlights that the power consumption decreases as both frequency and coverage increase (Figure 6.13b). Specifically, the expected power consumption is 5,280.4 W in the less intrusive configuration ($cov = 0.5$ and $int = 120$ minutes), while it is reduced to 5,074.2 W (as when no fault tolerance is adopted) if checkpointing all the jobs every 5 minutes with coverage equals to 1.0. The more the checkpoint is frequent and with high coverage, the more the fault activation is tolerated, thus improving the availability

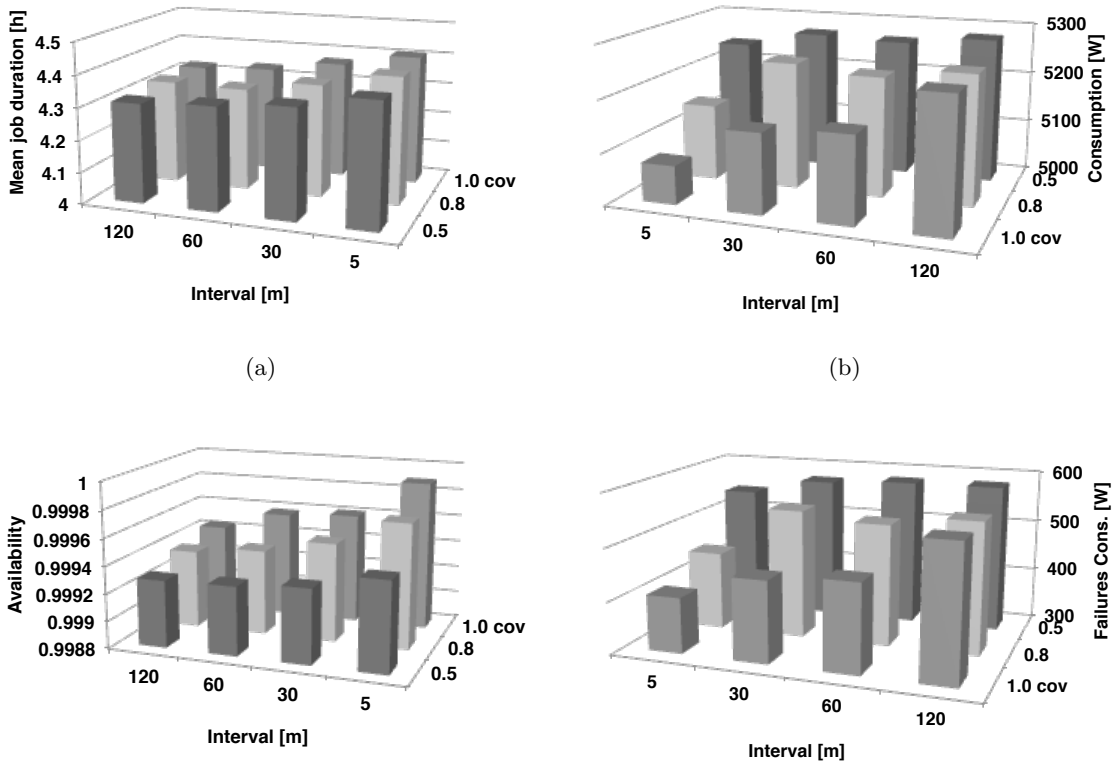


Figure 6.13: Results when varying the checkpointing coverage and frequency.

and reducing the fraction of energy wasted for failures (see Figure 6.13d). This shows that *job checkpointing is a consumable fault tolerance technique, since able to reinvest the energy saved for reduced failures in improving the fault tolerance itself* by increasing coverage and frequency. Also, performance is not significantly affected by the different tunings of the checkpointing. In particular, the variation of the mean job duration when varying the coverage and frequency of checkpointing is not statistically significant. Comparing the best configuration ($cov = 1$ and $int = 5$ minutes) with the basic one with 8 small VMs without fault tolerance discussed in Section 6.3.1, we can observe an increase of the availability from

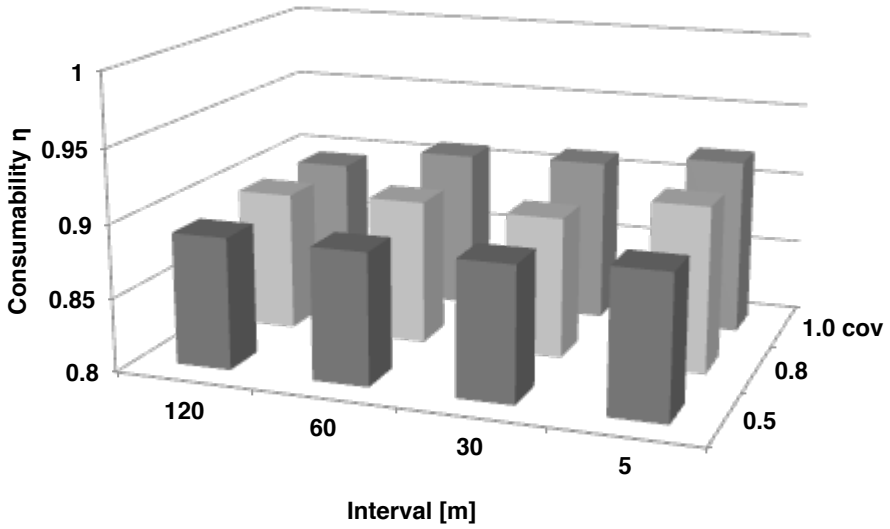


Figure 6.14: Consumability η when varying the checkpointing coverage and frequency.

0.99661 up to 0.99989, a reduction of the power consumption by 55%, and a performance increase by 67%.

The trend of the consumability metric η, depicted in Figure 6.14, confirms the capabilities of checkpointing, being it the opposite of what estimated for replication. It increases when improving the fault tolerance technique in terms of coverage and frequency. η is 88% when performing checkpointing of half of the jobs every 2 hours, and increases up to 92% when checkpointing all the jobs every 5 minutes.

6.4 Discussion

The case study presented in this Chapter provided a further proof of the importance of the *consumability* as an attribute of processing systems.

The consumability trend allows administrators to identify choices that increase the efficiency of the system from an engineering point of view. Cost-benefit analysis of management strategies for a processing system can neither overlook *any* of performance, consumption and dependability aspects, nor their mutual effects. Considering the three aspects separately or in couple, by means of performability and common energy efficiency analysis, would lead to improper choices. Moreover, the capabilities of the various management techniques would be hidden. As an instance, the scheduling of just one VM per PM may improve the performance-consumption ratio, while worsening the availability of the system.

It was demonstrated the effectiveness of the proposed solution for analyzing processing systems by showing how to evaluate tunings and configurations to reach the desired trade-off among performance, consumption and availability. Virtualization and a cloud platform can certainly help in the management of a data center for reducing energy consumption and implement fault tolerance, but we wanted to go beyond.

We showed how to assess the effect of virtualization on a batch processing system in terms of mean job duration, power consumption and availability. This study, apart from demonstrating the importance of consumability, acted as an additional example of how to perform the analysis.

For the virtualized S.Co.P.E. processing system, we showed that the wake up time necessary for using stand-by machines is negligible with respect to the mean job duration. On the contrary, the number of VMs that can run on each server of the system is a key factor to balance elasticity and isolation against performance decrease. In our setting, the use of 4 VMs per server provided the best balancing among elasticity and performance. This also

improved availability and consumption, since reducing the expected duration of the jobs. *The scheduling of VMs running different workloads on the same PM also appeared as an useful means for improving the system consumability.* To enhance availability, checkpointing and replication have been considered. *Job checkpointing looked like a consumable choice since the energy saved for the reduced number of failures is similar to the one necessary for implementing the fault tolerance itself.* With the right tuning, the system reached a consumability of 92%.

Conclusion

Reducing the energy consumption is the new agenda for owners and administrators of processing systems. The huge electricity requirements and the consequent carbon footprint can not be neglected anymore. It is an ethical question, a business question, and soon it will be a legal question, too. Design and management strategies have to cope with consumption constraints, apart from performance needs. That is, electricity must not be wasted. Failures appear as one of the main causes of electricity squandering and they are frequent in large scale systems. Carrying out an effective analysis of costs and benefits for various design and management strategies is then of paramount importance to find the best trade-off among performance, consumption and dependability aspects.

In this dissertation we focused on the efficiency of batch processing systems. The objective was both to demonstrate the mutual relations among the aspects of performance, consumption and dependability, gathered in the *consumability* attribute, and to solve the problem of system administrators about performing an analysis in order to reduce consumption without affecting performance and dependability or, in the other way around, to improve performance and/or availability avoiding to move consumption.

As for the current scientific literature, this dissertation contributed to the identified

needs and problems with:

- **The introduction of the *consumability* aspect of processing systems**, in order to deal with performance, consumption and dependability simultaneously. The evaluation of the energy efficiency can not disregard failures that commonly happen in large data centers, since they represent the wasted portion of the incoming power not transformed in useful power output, from a pure engineering point of view.

- **A systematic procedure to carry out the *consumability analysis*** of batch processing systems, which starts from the characterization of the system to populate a stochastic model that provides expectations of performance, total consumption, consumption due to failures, consumption due to fault tolerance (if implemented) and availability trends under various configurations. The analysis aims to support administrators in selecting configurations (e.g., scheduling algorithm or fault tolerance technique) and tunings (e.g., of a selected fault tolerance technique) that balance performance, consumption and availability with respect to organization needs.

- **The quantification of consumability and energy efficiency of processing systems** by means of a new metric that considers the ratio between the portion of the power actually used for producing useful work and the total input power. The metric includes performance, consumption and dependability information in a unified measure.

The solution proposed in this dissertation has been conceived for a generic batch processing system to support the estimation process independently from the specific system

hardware, software and existing configurations and load. Apart from the figures characterizing the *status quo* of the system, the consumability analysis can be exploited for other systems as well as for the presented case studies. Those studies were described step-by-step, with the included analysis of workload, performance, consumption, failure and adopted statistical techniques, also acting as a set of guidelines for the consumability analysis of other systems.

The two case studies, which are based on a real batch processing system, also demonstrated the importance of the consumability and the feasibility of the proposed solution. The experimental results confirmed that consumability is representative of system efficiency and effectiveness showing the importance of evaluating dependability jointly with performance and consumption for energy efficient, or better *consumable*, choices. The analysis revealed useful hints for the system administration (i) on the implementation of the fault tolerance strategy, since job replication appeared costly, even if it largely improves availability and keeps performance unchanged, and (ii) on the management of virtualized data centers, to avoid that performance and availability may worsen, even though the energy consumption is reduced.

Processing systems and related techniques and technologies should be evaluated on the basis of their consumability. As an instance, scheduling and load balancing strategies should be assessed by taking into account also their possible effects on availability. Dependability means should be assessed for their impact on consumption, instead. This thesis aims to serve as a foundation for future work on unified evaluation of performance, consumption

and dependability attributes.

It can be created a tool exploiting the proposed models to support administrative tasks. If the system is properly instrumented, data can be collected from the running system and aggregated to provide model inputs. The solution of the model provides the information for choices focused on the satisfaction of both functional and non-functional properties. Examples of such choices are system design improvement, hardware updates, tuning of the scheduling, workload balancing policies. The continuous monitoring, consumability analysis and tuning permit administrators to satisfy service specific goals with a reduced cost, in terms of energy consumption.

The benchmarking of processing systems should be revisited as well. The evaluation of the performance-consumption ratio in a controlled environment considers an ideal system and is not useful for practical purpose. By forcing failures, one can evaluate the system behavior in an extended set of conditions, including the common, faulty ones. Merging the approaches on performance-consumption benchmarking with those on dependability benchmarking would help to figure out the energy the system needs to perform at a certain level, the cost of the failures, the benefit of implementing fault tolerance, the efficiency in case of failures, that is *how much the system is consumable*.

Some of the contributions of this dissertation have been published in the following journals or conference proceedings:

- R. Ghosh, F. Longo, F. Frattini, S. Russo, K.S. Trivedi. *Scalable Analytics for IaaS Cloud Availability.* IEEE Transactions on Cloud Computing, vol. PP, no. 99, IEEE, 2014 (*to appear*).

- S. Sarkar, R. Ganesan, M. Cinque, F. Frattini, S. Russo, A. Savignano. *Mining Invariants from SaaS Application Logs.* Proc. of EDCC 2014 - 10th European Dependable Computing Conference, IEEE, 2014 (*to appear*).

- D. Cotroneo, F. Frattini, R. Natella, R. Pietrantuono. *Performance Degradation Analysis of a Supercomputer.* Proc. of IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW 2013), IEEE, 2013.

- M. Cinque, D. Cotroneo, F. Frattini, S. Russo. *Cost-Benefit Analysis of Virtualizing Batch Systems: Performance-Energy-Dependability Trade-offs.* Proc. of 6th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2013), IEEE/ACM, 2013.

- F. Frattini, R. Gosh, M. Cinque, A. Rindos, K.S. Trivedi. *Analysis of Bugs in Apache Virtual Computing Lab.* Proc. of 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2013), IEEE, 2013.

- F. Frattini, A. Bovenzi, J. Alonso, K.S. Trivedi. *Reliability Indices.* Encyclopedia of Operations Research and Management Science, Wiley, 2013.

Other published research papers by the author are:

- F. Frattini, M. Esposito, G. De Pietro. *MobiFuzzy: A Fuzzy Library to Build Mobile DSSs for Remote Patient Monitoring.* In M. Kamel, F. Karray, H. Hagras, Autonomous and Intelligent Systems, Lecture Notes in Computer Science, Springer, 2012.

- M. Cinque, F. Frattini, C. Esposito. *Leveraging Power of Transmission for Range-free Localization of Tiny Sensors.* In S. Martino, A. Peron, T. Tezuka, Web and Wireless Geographical Information Systems, Lecture Notes in Computer Science, Springer, 2012.

- F. Frattini, C. Esposito, S. Russo. *ROCRSSI++: an Efficient Localization Algorithm for Wireless Sensor Networks.* International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS), vol. 2, no. 2, IGI, 2011.

- F. Frattini, C. Esposito, S. Russo. *Efficient Range-free RSSI Localization Technique for Wireless Sensor Networks.* Proc. of 7th International Conference on Pervasive Services (ICPS 2010) - 4th International Workshop on Adaptive and DependAble Mobile Ubiquitous Systems (ADAMUS 2010), ACM, 2010.

Bibliography

[1] Adaptive Computing. Torque Resource Manager, mar 2014. http://www.adaptivecomputing.com.

[2] Miriam Allalouf, Yuriy Arbitman, Michael Factor, Ronen I. Kat, Kalman Meth, and Dalit Naor. Storage modeling for power estimation. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, SYSTOR '09. ACM, 2009.

[3] Mark Anderson. Better benchmarking for supercomputers. *Spectrum, IEEE*, 48(1):12–14, 2011.

[4] Apache. Apache VCL, mar 2014. https://vcl.apache.org.

[5] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33, 2004.

[6] Rami Bahsoon. A framework for dynamic self-optimization of power and dependability requirements in green cloud architectures. In *Proceedings of the 4th European conference on Software architecture*, pages 510–514. Springer-Verlag, 2010.

[7] J. Baliga, R. Ayre, K. Hinton, and R.S. Tucker. Photonic switching and the energy bottleneck. In *Photonics in Switching, 2007*, pages 125–126, Aug 2007.

[8] M.D. Beaudry. Performance-related reliability measures for computing systems. *Computers, IEEE Transactions on*, C-27(6):540–547, June 1978.

[9] Josep Ll. Berral, Íñigo Goiri, Ramón Nou, Ferran Julià, Jordi Guitart, Ricard Gavaldà, and Jordi Torres. Towards energy-aware scheduling in data centers using machine learning. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, 2010.

[10] R. Bertran, M. Gonzalez Tallada, X. Martorell, N. Navarro, and E. Ayguade. A systematic methodology to generate decomposable and responsive power models for cmps. *Computers, IEEE Transactions on*, PP(99):1–1, 2012.

[11] Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor S. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. John Wiley, 2006.

[12] R. Bolla, R. Bruschi, Franco Davoli, and F. Cucchietti. Energy efficiency in the future internet: A survey of existing approaches and trends in energy-aware fixed network infrastructures. *Communications Surveys Tutorials, IEEE*, 13(2):223–244, 2011.

[13] Tom Bostoen, Sape Mullender, and Yolande Berbers. Power-reduction techniques for data-center storage systems. *ACM Comput. Surv.*, 45(3):33:1–33:38, July 2013.

[14] M. Bouguerra, D. Trystram, and F. Wagner. Complexity analysis of checkpoint scheduling with variable costs. *IEEE Transactions on Computers*, PP(99):1–1, 2012.

[15] G. Callou, P. Maciel, D. Tutsch, and J. Araujo. Models for dependability and sustainability analysis of data center cooling architectures. In *Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on*, pages 1–6, 2012.

[16] M. Calzarossa and G. Serazzi. A characterization of the variation in time of workload arrival patterns. *Computers, IEEE Transactions on*, C-34(2):156–162, 1985.

[17] M. Calzarossa and G. Serazzi. Workload characterization: a survey. *Proceedings of the IEEE*, 81(8):1136–1150, Aug 1993.

[18] Maria Calzarossa, Luisa Massari, and Daniele Tessera. Workload characterization issues and methodologies. In *Performance Evaluation: Origins and Directions*, pages 459–481. Springer-Verlag, 2000.

[19] CERN. ATLAS Experiment, mar 2013. http://atlas.ch.

[20] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, and S. Wright. Power awareness in network design and routing. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, April 2008.

[21] N.S. Chauhan and A. Saxena. A green software development life cycle for cloud computing. *IT Professional*, 15(1):28–34, Jan 2013.

[22] Xin Chen, Qiang Liu, and JianBing Lai. A new power-aware scheduling algorithm for distributed system. In *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, Dec 2010.

[23] Jeonghwan Choi, S. Govindan, Jinkyu Jeong, B. Urgaonkar, and Anand Sivasubramaniam. Power consumption prediction and power-aware packing in consolidated environments. *Computers, IEEE Transactions on*, 59(12):1640–1654, Dec 2010.

[24] M. Chtepen, F.H.A. Claeys, B. Dhoedt, F. De Turck, P. Demeester, and P.A. Vanrolleghem. Adaptive task checkpointing and replication: Toward efficient fault-tolerant grids. *IEEE Transactions on Parallel and Distributed Systems*, 20(2):180–190, 2009.

[25] Wu chun Feng and K.W. Cameron. The green500 list: Encouraging sustainable supercomputing. *Computer*, 40(12):50–55, Dec 2007.

[26] G. Ciardo, A. Blakemore, P. F. Chimento, J.K. Muppala, and K.S. Trivedi. Automated generation and analysis of markov reward models using stochastic reward nets. *IMA Volumes in Mathematics and its Applications: Linear Algebra, Markov Chains, and Queueing Models - Meyer, C.; Plemmons, R.J*, 48:145–191, 1993.

[27] Cienca. HPC for Science, mar 2014. http://www.hpc.cineca.it/content/hpc-science.

[28] M. Cinque, D. Cotroneo, F. Frattini, and S. Russo. Cost-benefit analysis of virtualizing batch systems: Performance-energy-dependability trade-offs. In *Proc. of 6th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, 2013.

[29] The European Commission. Commission regulation (eu) no 617/2013. *Official Journal of the European Union*, June 2013.

[30] D. Cotroneo, F. Frattini, R. Natella, and R. Pietrantuono. Performance degradation analysis of a supercomputer. In *Proc. of IEEE 23rd International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2013.

[31] D. Anderson. A Framework for Data Center Energy Productivity, April 2008. The Green Grid, White paper.

[32] D. de Andres, J.-C. Ruiz, and P. Gil. Using dependability, performance, area and energy consumption experimental measures to benchmark ip cores. In *Dependable Computing, 2009. LADC '09. Fourth Latin-American Symposium on*, pages 49–56, 2009.

[33] Yuhui Deng. What is the future of disk drives, death or rebirth? *ACM Comput. Surv.*, 43(3), April 2011.

[34] C. Di Martino, M. Cinque, and D. Cotroneo. Assessing time coalescence techniques for the analysis of supercomputer logs. In *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*, pages 1–12, June 2012.

[35] C. Di Martino, M. Cinque, and D. Cotroneo. Assessing time coalescence techniques for the analysis of supercomputer logs. In *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*, pages 1–12, 2012.

[36] Catello Di Martino, Marcello Cinque, and Domenico Cotroneo. Automated generation of performance and dependability models for the assessment of wireless sensor networks. *IEEE Trans. Comput.*, 61(6):870–884, June 2012.

[37] M. Dick, J. Drangmeister, E. Kern, and S. Naumann. Green software engineering with agile methods. In *Green and Sustainable Software (GREENS), 2013 2nd International Workshop on*, pages 78–85, May 2013.

[38] E. Samadiani, Y. Joshi, J.K. Allen and F. Mistree. Adaptable robust design of multi-scale convective systems applied to energy efficient data centers. *Numerical Heat Transfer, Part A: Applications: An International Journal of Computation and Methodology*, 57(2), 2010.

[39] A. Ejlali, B.M. Al-Hashimi, P. Rosinger, and S.G. Miremadi. Joint consideration of fault-tolerance, energy-efficiency and performance in on-chip networks. In *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, pages 1–6, 2007.

[40] Nosayba El-Sayed, Ioan A. Stefanovici, George Amvrosiadis, Andy A. Hwang, and Bianca Schroeder. Temperature management in data centers: Why some (might) like it hot. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '12. ACM, 2012.

[41] Energy Conservation Center. Top Runner program, mar 2014. http://www.eccj.or.jp/top_runner.

[42] Environmental Protection Agency. About Energy Star, mar 2014. https://www.energystar.gov/about.

[43] Ericsson. Energy and carbon report, feb 2014. http://www.ericsson.com/res/docs/2013/ericsson-energy-and-carbon-report.pdf.

[44] European Commission DG Information Society. ICT Footprint, mar 2014. http://www.ict-footprint.com.

[45] Wu-chun Feng. Making a case for efficient supercomputing. *Queue*, 1(7):54–64, October 2003.

[46] Xizhou Feng, Rong Ge, and K.W. Cameron. Power and energy profiling of scientific applications on distributed systems. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 34–34, 2005.

[47] F. Frattini, R. Ghosh, M. Cinque, A. Rindos, and K.S. Trivedi. Analysis of bugs in apache virtual computing lab. In *Proc of IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2013.

[48] Flavio Frattini, Antonio Bovenzi, Javier Alonso, and Kishor Trivedi. Reliability indices. *Wiley Encyclopedia of Operations Research and Management Science*, 2010.

[49] A. Gandhi, Yuan Chen, D. Gmach, M. Arlitt, and M. Marwah. Minimizing data center sla violations and power consumption via hybrid resource provisioning. In *Proceedings of the 2011 International Green Computing Conference and Workshops*, pages 1–8. IEEE Computer Society, 2011.

[50] Rong Ge, Xizhou Feng, and K.W. Cameron. Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, pages 34–34, Nov 2005.

[51] Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and K.W. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Transactions on Parallel and Distributed Systems*, 21(5):658 –671, may 2010.

[52] Kanad Ghose and Milind B. Kamble. Reducing power in superscalar processor caches using subbanking, multiple line buffers and bit-line segmentation. In *Proceedings of the 1999 International Symposium on Low Power Electronics and Design*, ISLPED '99, 1999.

[53] R. Ghosh, F. Longo, F. Frattini, S. Russo, and K.S. Trivedi. Scalable analytics for iaas cloud availability. *Cloud Computing, IEEE Transactions on*, PP(99):1–1, 2014.

[54] R. Ghosh, V.K. Naik, and K.S. Trivedi. Power-performance trade-offs in iaas cloud: A scalable analytic approach. In *Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on*, pages 152–157, 2011.

[55] G.L. Valentini et al. An overview of energy efficiency techniques in cluster computing systems. *Cluster Computing*, 16(1):3–15, 2013.

[56] I. Goiri, Kien Le, M.E. Haque, R. Beauchea, T.D. Nguyen, J. Guitart, J. Torres, and R. Bianchini. Greenslot: Scheduling energy consumption in green datacenters. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 1–11, 2011.

[57] Pablo Graubner, Matthias Schmidt, and Bernd Freisleben. Energy-efficient virtual machine consolidation. *IT Professional*, 15(2):28–34, March 2013.

[58] Green 500 project. The Green 500, mar 2013. http://www.green500.org.

[59] D. Harris. *Reducing FPGA Power Consumption.* Penton Media Inc., 2008.

[60] J. Hedman and S. Henningsson. Three strategies for green it. *IT Professional*, 13(1):54–57, Jan 2011.

[61] Eric Heien, Derrick Kondo, Ana Gainaru, Dan LaPine, Bill Kramer, and Franck Cappello. Modeling and tolerating heterogeneous failures in large parallel systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 45:1–45:11, 2011.

[62] C. Hirel, B. Tuffin, and K. S. Trivedi. SPNP: Stochastic Petri Nets. Version 6. In Springer Verlag, editor, *Computer Performance Evaluation: Modelling Techniques and Tools*, volume 1768, 2000.

[63] H. HOLLMANN. A relation between levenshtein-type distances and insertion-and-deletion correcting capabilities of codes. *Information Theory, IEEE Transactions on*, 39(4):1424–1427, Jul 1993.

[64] M.M. Hossain, Jen-Cheng Huang, and H.-H.S. Lee. Migration energy-aware workload consolidation in enterprise clouds. In *Proc. of IEEE CloudCom*, 2012.

[65] J.S. Hu, N. Vijaykrishnan, M.J. Irwin, and M. Kandemir. Using dynamic branch behavior for power-efficient instruction fetch. In *VLSI, 2003. Proceedings. IEEE Computer Society Annual Symposium on*, Feb 2003.

[66] I. Krsul et al. Vmplants: Providing and managing virtual machine execution environments for grid computing. In *Proc. of ACM/IEEE Conference on Supercomputing (SC)*, 2004.

[67] IGI - Italian Grid Infrastructure. Troubleshooting guide for CREAM, apr 2013. https://wiki.italiangrid.it/twiki/bin/view/CREAM/ TroubleshootingGuide.

[68] International Telecommunications Union. Recommendation E.800, mar 2014. http://www.itu.int/rec/T-REC-E.800/en.

[69] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* Wiley-Interscience, New York, NY, 1991.

[70] J.W.Tukey. *Exploratory data analysis.* Addison-Wesely, 1977.

[71] Z. Kalbarczyk. Measurement-based analysis: A key to experimental research in dependability. In *Dependable Computing Conference, 2006. EDCC '06. Sixth European*, Oct 2006.

[72] Karama Kanoun and Lisa Spainhower. *Dependability Benchmarking for Computer Systems.* Wiley-IEEE Computer Society Press, 2008.

[73] A. Kennedy, Xiaojun Wang, and Bin Liu. Energy efficient packet classification hardware accelerator. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, 2008.

[74] Alan Kennedy, Xiaojun Wang, Zhen Liu, and Bin Liu. Low power architecture for high speed packet classification. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '08, 2008.

[75] J. Kin, M. Gupta, and W.H. Mangione-Smith. The filter cache: an energy efficient memory structure. In *Microarchitecture, 1997. Proceedings., Thirtieth Annual IEEE/ACM International Symposium on*, Dec 1997.

[76] Thomas Kistler and Michael Franz. Continuous program optimization: A case study. *ACM Trans. Program. Lang. Syst.*, 25(4), July 2003.

[77] M.V. Konshak. Modular liquid cooling of electronic components while preserving data center integrity, December 27 2007. US Patent App. 11/426,238.

[78] Umesh Krishnaswamy and Isaac D. Scherson. A framework for computer performance evaluation using benchmark sets. *IEEE Trans. Comput.*, 49(12):1325–1338, December 2000.

[79] Patrick Kurp. Green computing. *Commun. ACM*, 51(10):11–13, October 2008.

[80] EunKyung Lee, Indraneel Kulkarni, Dario Pompili, and Manish Parashar. Proactive thermal management in green datacenters. *The Journal of Supercomputing*, 60(2):165–195, 2012.

[81] Sangmin Lee, Rina Panigrahy, Vijayan Prabhakaran, and Venugopalan Ramasubramanian. Validating heuristics for virtual machines consolidation. *Microsoft Research - Technical Report*, 2011.

[82] YoungChoon Lee and AlbertY. Zomaya. Energy efficient utilization of resources in cloud computing systems. *The Journal of Supercomputing*, 60(2):268–280, 2012.

[83] Keqin Li. Energy efficient scheduling of parallel tasks onÂ multiprocessor computers. *The Journal of Supercomputing*, 60(2):223–247, 2012.

[84] David J. Lilja. *Measurring Computer Performance. A practitioners guide.* Cambridge University Press, 2004.

[85] Yan Luo, Jia Yu, Jun Yang, and Laxmi N. Bhuyan. Conserving network processor power consumption by exploiting traffic variability. *ACM Trans. Archit. Code Optim.*, 4(1), March 2007.

[86] C. Mastroianni, M. Meo, and G. Papuzzo. Probabilistic consolidation of virtual machines in self-organizing cloud data centers. *Cloud Computing, IEEE Transactions on*, 1(2):215–228, July 2013.

[87] Mathew P., Ganguly S., Greenberg S., Sartor D. Self-benchmarking guide for data centers: metrics, benchmarks, actions, July 2009. Lawrence Berkeley National Laboratory, Technical report.

[88] Esteban Meneses, Osman Sarood, and Laxmikant V. Kale. Assessing energy efficiency of fault tolerance protocols for hpc systems. In *Proceedings of the 2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing*, SBAC-PAD '12, 2012.

[89] J.F. Meyer. On evaluating the performability of degradable computing systems. *Computers, IEEE Transactions on*, C-29(8):720–731, 1980.

[90] Denis Sheahan Sun Microsystems. UltraSPARC T1 and T2 performance, mar 2014. https://blogs.oracle.com/deniss/entry/ultrasparc_t1_low_power_and1.

[91] Adam Moody, Greg Bronevetsky, Kathryn Mohror, and Bronis R. de Supinski. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–11, 2010.

[92] Tridib Mukherjee, Ayan Banerjee, Georgios Varsamopoulos, Sandeep K.S. Gupta, and Sanjay Rungta. Spatio-temporal thermal-aware job scheduling to minimize energy consumption in virtualized heterogeneous data centers. *Computer Networks*, 53(17):2888 – 2904, 2009.

[93] San Murugesan. Harnessing green it: Principles and practices. *IT Professional*, 10(1):24–33, Jan 2008.

[94] V.K. Naik, S.K. Setia, and M.S. Squillante. Performance analysis of job scheduling policies in parallel supercomputing environments. In *Supercomputing '93. Proceedings*, pages 824–833, 1993.

[95] NCSA. Batch jobs - sample script, mar 2014. https://bluewaters.ncsa.illinois.edu/batch-jobs.

[96] B. Nicolae and F. Cappello. Blobcr: Efficient checkpoint-restart for hpc applications on iaas clouds using virtual disk image snapshots. In *Proc. of ACM SC*, 2011.

[97] NIST - National Institute of Standards and Technology, Peter Mell, and Timothy Grance. The NIST definition of cloud computing. *Special Publication 800-145*, 2011.

[98] Ohio Supercomputer Center. Batch Processing at OSC, mar 2014. https://www.osc.edu/supercomputing/batch-processing-at-osc.

[99] A. Oliner and J. Stearley. What supercomputers say: A study of five system logs. In *Dependable Systems and Networks, 2007. DSN '07. 37th Annual IEEE/IFIP International Conference on*, pages 575–584, 2007.

[100] A. J. Oliner, R. K. Sahoo, J. E. Moreira, and M. Gupta. Performance implications of periodic checkpointing on large-scale cluster systems. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 18 - Volume 19*, IPDPS '05, pages 299.2–, Washington, DC, USA, 2005. IEEE Computer Society.

[101] A.-C. Orgerie, L. Lefevre, and J.-P. Gelas. Demystifying energy consumption in grids and clouds. In *Green Computing Conference, 2010 International*, pages 335–342, 2010.

[102] Ehsan Pakbaznia and Massoud Pedram. Minimizing data center cooling and server power costs. In *Proceedings of the 14th ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '09, 2009.

[103] Jeffrey Palm, Han Lee, Amer Diwan, and J. Eliot B. Moss. When to use a compilation service? *SIGPLAN Not.*, 37(7), June 2002.

[104] C.D. Patel, C.E. Bash, and A.H. Beitelmal. Smart cooling of data centers, June 3 2003. US Patent 6,574,104.

[105] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, July 1974.

[106] Diego Reforgiato Recupero. Toward a green internet. *Science*, 339:1533–1534, 2013.

[107] A. Rezaei, H. Salimi, and M. Sharifi. Improving software dependability using system-level virtualization: A survey. In *Proc. of WAINA*, 2010.

[108] Rice University - Division of Information Technology. Why Are My Jobs Not Running?, apr 2013. http://rcsg.rice.edu/rcsg/shared/scheduling.html.

[109] F. Ryckbosch, S. Polfliet, and L. Eeckhout. Trends in server energy proportionality. *Computer*, 44(9):69–72, 2011.

[110] M. Sabharwal, A. Agrawal, and G. Metri. Enabling green it through energy-aware software. *IT Professional*, 15(1):19–27, Jan 2013.

[111] M.R. Sabharwal. Software power optimization: Analysis and optimization for energy-efficient software. In *Low Power Electronics and Design (ISLPED) 2011 International Symposium on*, pages 63–64, Aug 2011.

[112] R.A. Sahner, K.S. Trivedi, and A. Puliafito. *Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software*. Kluwer Academic Publishers, 1996.

[113] T.K. Samuel, T. Baer, R.G. Brook, M. Ezell, and P. Kovatch. Scheduling diverse high performance computing systems with the goal of maximizing utilization. In *High Performance Computing (HiPC), 2011 18th International Conference on*, 2011.

[114] WilliamH. Sanders and JohnF. Meyer. Stochastic activity networks: Formal definitions and concepts. In Ed Brinksma, Holger Hermanns, and Joost-Pieter Katoen, editors, *Lectures on Formal Methods and PerformanceAnalysis*, volume 2090 of *Lecture Notes in Computer Science*, pages 315–343. Springer Berlin Heidelberg, 2001.

[115] S. Sarkar, R. Ganesan, M. Cinque, F. Frattini, S. Russo, and A. Savignano. Mining invariants from saas application logs. In *Dependable Computing Conference (EDCC), 2014 Tenth European*, pages 1–8, May 2014.

[116] Osman Sarood and Laxmikant V. Kale. A 'cool' load balancer for parallel applications. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 21:1–21:11, New York, NY, USA, 2011. ACM.

[117] T. Sato and T. Funaki. Dependability, power, and performance trade-off on a multicore processor. In *Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific*, pages 714–719, 2008.

[118] Bianca Schroeder and Garth A. Gibson. Understanding disk failure rates: What does an mttf of 1,000,000 hours mean to you? *Trans. Storage*, 3(3), 2007.

[119] D.J. Schumacher and W.C. Beckman. Data center cooling system, April 23 2002. US Patent 6,374,627.

[120] S. Sharma, C.-H. Hsu, and Wu chun Feng. Making a case for a green500 list. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, 2006.

[121] Anand Sivasubramaniam, Mahmut T. Kandemir, Narayanan Vijaykrishnan, and Mary Jane Irwin. Designing energy-efficient software. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, IPDPS '02, 2002.

[122] SPEC - Standard Performance Evaluation Corporation. SPECpower_ssj2008 User Guide, apr 2013. http://www.spec.org/power/docs/ SPECpower_ssj2008-User_Guide.pdf.

[123] B. Subramaniam and W.-C. Feng. Gbench: benchmarking methodology for evaluating the energy efficiency of supercomputers. *Computer Science - Research and Development.*, pages 1–10, 2012.

[124] Balaji Subramaniam and Wu-chun Feng. Statistical power and performance modeling for optimizing the energy efficiency of scientific computing. In *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, pages 139–146. IEEE Computer Society, 2010.

[125] T. Tan, A. Raghunathan, N. Jha. Software architectural transformations: A new approach to low energy embedded software. In *Design, Automation and Test in Europe*, 2003.

[126] The European Commission. Intelligent Energy Europe, mar 2014. http://ec.europa.eu/energy/intelligent.

[127] Top 500 project. Top500 Supercomputers Site, mar 2013. http://www.top500.org.

[128] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications, second edition*. Wiley, 2001.

[129] Università degli Studi di Napoli Federico II. Scope, mar 2014. http://www.scope.unina.it.

[130] University of California, Berkeley. Ganglia Monitoring System, mar 2014. http://ganglia.info.

[131] Vasanth Venkatachalam and Michael Franz. Power reduction techniques for microprocessor systems. *ACM Comput. Surv.*, 37(3):195–237, September 2005.

[132] Akshat Verma, Puneet Ahuja, and Anindya Neogi. Power-aware dynamic placement of hpc applications. In *Proceedings of the 22nd annual international conference on Supercomputing*, ICS '08, pages 175–184, 2008.

[133] W. Deng et al. Lifetime or energy: Consolidating servers with reliability control in virtualized cloud datacenters. In *Proc. of IEEE CloudCom*, 2012.

[134] Dazhi Wang and KishorS. Trivedi. Modeling user-perceived service availability. In Miroslaw Malek, Edgar Nett, and Neeraj Suri, editors, *Service Availability*, volume 3694 of *Lecture Notes in Computer Science*, pages 107–122. Springer Berlin Heidelberg, 2005.

[135] Dazhi Wang, Wei Xie, and Kishor S. Trivedi. Performability analysis of clustered systems with rejuvenation under varying workload. *Perform. Eval.*, 64(3):247–265, 2007.

[136] Lizhe Wang, SameeU. Khan, and Jai Dayal. Thermal aware workload placement with task-temperature profiles in a data center. *The Journal of Supercomputing*, 61(3):780–803, 2012.

[137] Meng Wang, Xiaoqiao Meng, and Li Zhang. Consolidating virtual machines with dynamic bandwidth demand in data centers. In *INFOCOM, 2011 Proceedings IEEE*, pages 71–75, 2011.

[138] I. Witten, E. Frank, and M. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Elsevier, 3rd edition, 2011.

[139] Priyanka Yadav, Vineet Khanna, and Deepak Singh Rawat. Stochastic performance modeling of manets implementing aodv under hidden-exposed terminal problem. *International Journal of Computer Science & Engineering Technology (IJCSET)*, 5(1):31–45, Jan 2014.

[140] M. Yamada, T. Yazaki, N. Matsuyama, and T. Hayashi. Power efficient approach and performance control for routers. In *Communications Workshops, 2009. ICC Workshops 2009. IEEE International Conference on*, 2009.

[141] Qian Zhu, Jiedan Zhu, and G. Agrawal. Power-aware consolidation of scientific workflows in virtualized environments. In *Proc. of ACM SC*, 2010.

[142] Stylianos Zikos and Helen D. Karatza. Performance and energy aware cluster-level scheduling of compute-intensive jobs with unknown service times. *Simulation Modelling Practice and Theory*, 19(1):239 – 250, 2011.