
**POLITECNICO DI MILANO**
**DIPARTIMENTO DI ELETTRONICA, INFORMAZIONE E BIOINGEGNERIA**
**DOCTORAL PROGRAM IN INFORMATION TECHNOLOGY**

---

# ENERGY EFFICIENCY IN LARGE DATA-CENTERS USING PERFORMANCE EVALUATION TECHNIQUES

Doctoral Dissertation of:
**Riccardo Pinciroli**

Advisors:
**Prof. Marco Gribaudo**

Co-advisors:
**Prof. Giuseppe Serazzi**

Tutor:
**Prof. Barbara Pernici**

The Chair of the Doctoral Program:
**Prof. Andrea Bonarini**

XXX Cycle

# Acknowledgments

I would like to thank my mentors, Prof. Marco Gribaudo and Prof. Giuseppe Serazzi, who supported me during these three years and made me love this amazing job. This thesis would have not been possible without them and their advice.

A special thank is for Prof. Kishor Trivedi and his research group DHAAL. Thank you for the interesting talks, for hosting me at Duke University and to conduct exciting research together.

I would also like to thank my colleagues and co-authors Prof. Fabio Antonelli, Prof. Andrea Bobbio, Prof. Vittorio Cortellessa, Prof. Salvatore Distefano, Davide Cerotti, Pietro Piazzolla and Catia Trubiani. It has been a pleasure time collaborating with you.

I am grateful to the thesis reviewers, Prof. Anne Remke and Daniele Manini, for their careful reviews and helpful comments. All their suggestions have been valuable for improving my dissertation.

I thank of course Marta and my friends for the time spent together. My life would not be the same without you all.

Finally, I want to say thank you to my family for always being there for me: to my parents for making this story possible, and to support me whenever I need it; to my brother for our interesting conversations.

# Abstract

Nowadays, the amount of available computing resources is limitless and the energy consumption in IT is assuming an increasing importance due to the huge number of compute-intensive applications used in business and scientific fields. In 2014 the IT infrastructures in United States consumed about 70 billion kWh and this data has been estimated to grow to 73 billion kWh by 2020. Besides the significant amounts of money (e.g., 7.4 billion dollars in 2011), large energy consumption values also come with enormous volume of greenhouse gases released into the atmosphere.

While several efforts have been done to improve the data-centers' energy efficiency, their energy consumption must be further reduced, for example, taking into consideration the workload heterogeneity. In this thesis, new techniques to evaluate and improve the energy efficiency of data-centers are proposed. To this end, performance modeling approaches such as queuing networks, Petri nets and stochastic processes are applied, and analytical and discrete event simulation techniques are used.

To investigate the data-centers energy consumption problem, four different paths are followed: *i)* a new power model is proposed for better characterize current systems' architecture; *ii)* a new energy metric is introduced to account for data-centers' workload heterogeneity; *iii)* a new framework is adopted to study Big data applications; *iv)* epistemic uncertainty is propagated into a model to improve the accuracy of its output measures.

The power model proposed in this thesis accounts for both *dynamic voltage/frequency scaling* and *simultaneous multi-threading*, two widespread

energy saving techniques used in multi-core architectures. The *energy per time-unit of execution* is the new energy-performance trade-off metric that also considers the heterogeneity of the workload. *Pool depletion systems* is the framework proposed to analyze all the applications that generate a large number of tasks which must be executed by one or more subsystems with limited capacity (e.g., Big Data applications), and investigate new scheduling strategies to decrease their execution time. Finally, since stochastic models are fundamental in this thesis, *epistemic uncertainty propagation* is studied. It allows the modeler to take into account the impact of uncertain input parameters on the output measures of the model.

# Sommario

Calcolatori più performanti e un maggior numero di operazioni complesse eseguite sia dalle industrie sia nel campo scientifico hanno portato a un elevato consumo energetico nel settore delle tecnologie dell'informazione. Negli Stati Uniti, tale consumo energetico ammontava a 70 TWh nel 2014 e lo si stima crescere fino a 73 TWh entro il 2020. Il consumo di grandi quantità di energia è un problema per il settore dell'informazione a causa degli ingenti costi economici e ambientali.

Nonostante siano stati fatti molti sforzi per diminuire il consumo energetico di queste infrastrutture, ancora molto può essere fatto tenendo in considerazione, per esempio, l'eterogeneità delle applicazioni eseguite. In questa tesi vengono proposti nuovi metodi per valutare e migliorare l'efficienza energetica dei data-center, ricorrendo ad approcci analitici e simulativi per la modellazione delle prestazioni, quali le reti di code, le reti di Petri e i processi stocastici. Per affrontare questo problema abbiamo fatto ricorso a diverse metodologie come: *i)* un nuovo modello per la miglior caratterizzazione dell'architettura e del consumo energetico dei nuovi sistemi; *ii)* una nuova metrica che considera l'eterogeneità del carico di lavoro di questi sistemi per valutare con più precisione le loro prestazioni; *iii)* un nuovo framework per studiare le applicazioni Big Data; *iv)* l'incertezza epistemica usata per migliorare l'accuratezza dei modelli.

Diversamente dai modelli di potenza attualmente disponibili, quello proposto in questa tesi tiene in considerazione, contemporaneamente, due tecniche per il risparmio energetico molto diffuse: il *dynamic voltage/frequency scaling* e il *simultaneous multi-threading*. L'*energia per unità di*

*tempo di esecuzione*, la nuova metrica proposta in questa tesi, valuta anche l'eterogeneità delle applicazioni per studiare il trade-off tra le performance del sistema e il suo consumo energetico. *Pool depletion systems* è il framework proposto sia per studiare tutte le applicazioni che generano un gran numero di richieste poi eseguite da uno o più sottosistemi con capacità limitata, sia per proporre una nuova strategia di pianificazione in grado di diminuire il tempo di esecuzione di queste applicazioni. Infine, poiché i modelli stocastici sono di fondamentale importanza in questa tesi, è stata considerata anche la *propagazione dell'incertezza epistemica*, in quanto permette di studiare come i risultati di un modello vengano influenzati dall'incertezza dei suoi parametri in ingresso.

# Contents

CHAPTER 1

# Introduction

## 1.1 Motivations: the need for energy efficiency

Data-centers power consumption is one of the greatest issues that IT organizations must face; indeed, to enable future expansions [8], it is necessary to take under control the energy related costs. The U.S. data-centers energy consumption will be 73 billion kWh in 2020 [124]. Although it is estimated to increase only by $4\%$ with respect to 2014, it is still an important problem, if considering that the worldwide amount of energy consumed for data-centers in 2014 has been estimated to be 270 TWh [136]. Moreover, even if large data-centers have made great improvements in energy efficiency, small and medium data-centers – that are generally used in medical, retail, office, and education sectors – are difficult to monitor and may affect the global energy consumption trend due to their wide diffusion [55].

Besides being a huge cost for every IT company, the energy high demand of data-centers has also an environmental impact due to the emission of large quantities of greenhouse gases. Indeed, according to the literature, data-centers are estimated to be responsible for $2\%$ to $10\%$ of global $CO_2$ emissions [96, 109] and the U.S. Environmental Protection Agency estimated that 67.9 million metric tons of $CO_2$ have been released in the

atmosphere in 2011 by IT industry [10].

For these reasons, the reduction of data-centers energy consumption is crucial when new infrastructures and applications are developed. Reducing energy consumption allows IT companies to increase their revenues in several different ways. First of all, it lowers the electricity bill, letting the company save several money. Then, reducing the carbon tax the organization must pay when the $CO_2$ emitted by their infrastructure exceeds some thresholds [116]. Finally, the governments may also apply some credit and cap-and-trade systems in order to limit the data-centers $CO_2$ production [85, 129]. For instance, when a business exceeds allowed carbon dioxide emission levels, it must purchase either credits or carbon offsets from the market – usually, less polluting companies – to be permitted to keep on working and producing $CO_2$.

## 1.2 Problems and research objectives

Due to its relevance for IT organizations, the problem of energy consumption in data-centers has been largely studied by scientists from industry and academia. Many different techniques have been proposed, such as frameworks and tools to directly reduce the energy consumption of a server, or scheduling strategies to better allocate the incoming requests, thus being able to turn off unused resources. Nonetheless, several efforts are still required in order to further improve the available techniques and introduce new approaches. The main problems studied in this thesis are:

- **Power models**. As said, over the years several techniques and strategies have been proposed in literature in order to improve the data-centers energy efficiency. However, most of the proposals require a suitable power model that can estimate the power consumption of the system starting from simple system parameters, such as the utilization of its resources. Especially when the techniques considered are based on results that exhibit a non-linear behavior, the accuracy of the power consumption model is of paramount importance to correctly identify the optimal system configurations that can achieve the target performance with the lowest possible power budget. The typical use of analytic expressions for the power computation is the inclusion of energy characterization in models of the system defined using suitable formalisms, i.e., queuing networks, Petri nets, Markov chains. Taking into account widespread commercial power reduction techniques, like dynamic voltage/frequency scaling (DVFS) and multi-threading, we can provide a more detailed power consumption expressions that can

be used to obtain more accurate estimates. Unfortunately, the available power models either do not consider these power-saving features, or do not account for both of them at the same time [16, 107, 149].

- **Multi-class workloads**. The available techniques and metrics proposed to reduce power consumption through the application of efficient load control strategies rarely take into consideration the multi-class nature of system workload, i.e., jobs with different characteristics and behavior. However, this type of workloads may be composed by a mix of jobs that saturates different resources, thus it is an important system's feature which may be exploited in order to increase the utilization of each resource, thus decreasing the idle period of a server. Indeed, a scheduler that accounts for the mix of requests in execution could make the system work with an optimal mix. This way, the utilization of each server increases and the system's energy consumption decreases, since a strong relationship between a server power consumption and its CPU utilization has been proven in literature [49, 138]. However, the available power metrics [8] analyze the power consumption of a data-center assuming it is processing only a single class of job. This assumption may decrease the accuracy of the power/energy metrics and, in the worst case, also make the user take a sub-optimal decision about the strategy that should be adopted.
- **Epistemic uncertainty**. When a model is adopted to study a physical system, its input parameters are usually estimated from either observations or experts' opinions. In both these cases, the value of the parameters is not the exact one since it is derived from a finite number of samples. For this reason, it is necessary to assume that some of the input parameters of the model are assessed with uncertainty, thus to be stochastic. Those input parameters become input random variables with a probability density function (pdf). Thus, the model does not return an exact value, but some stochastic results with a confidence interval. Differently from aleatory uncertainty – that has been largely considered in literature [58, 117] and that is due to the natural variations of the physical phenomenon modeled – epistemic uncertainty is introduced into the model by a lack of knowledge (i.e., finite number of observations) and needs to be propagated to the output. Albeit in the former case the uncertainty is reduced improving the model itself, in the latter one it may be curtailed by collecting a larger amount of samples for a more accurate input parameters estimation.
- **Other systems and applications**. Energy consumption has become

such an important problem also due to the wide diffusion of portable devices (e.g., smartphones, sensors, etc.) [10, 106]. Indeed, they are usually powered by a battery with a short lifetime, thus it is important to efficiently manage power and energy requirements in order to extend their life. In other words, reducing energy consumption is fundamental when dealing with mobile devices in order to increase the lifetime of their battery, differently from the data-centers case where energy efficiency is crucial to increase the companies revenues. For these reasons, analyzing energy consumption in small devices to make their lifetime longer is another important feature that may be enabled by power models.

## 1.3 Possible solutions and original contributions

In order to cope with the problems highlighted in Section 1.2, different models and techniques are proposed and applied in this thesis. They are presented and briefly discussed in the following.

- A model to estimate the power consumption of a multi-core CPU, when both DVFS and multi-threading are enabled, is provided. It needs some parameters that may be easily derived from operational and machine characteristics (e.g., number of active threads, frequency, etc.) and some calibration parameters that must be derived from the experiments. It has been tested considering different physical machines and its performance have been evaluated against those of the already available power models.
- In order to take into consideration the multi-class workload and its properties, two different strategies have been pursued:
  - a new energy metric, namely *energy per time-unit of execution* (EX), has been proposed to let the users consider different classes of request. This metric may be exploited to evaluate the performance of energy saving strategies in a multi-class environment;
  - *Pool depletion systems* are proposed and deeply analyzed. This new framework lets the users consider a system made of a pool of heterogeneous tasks that must be depleted in the shortest time by machines with finite capacity. Besides varying the capacity of each server and the number of tasks in the pool, this framework is also studied considering a different number of available servers and multi-core components.

- The epistemic uncertainty propagation is studied from different point of views. First of all, the case of an M/M/1 queue is analyzed assuming the uncertain input parameters are the inter-arrival and service rates. In particular, the average number of customer in the queue and its average response time are studied based on the distribution of the two input parameters. Then, epistemic uncertainty propagation is applied to the case of a power model; for this purpose, the input parameters that affect the most the output measures are identified and further analyzed. Furthermore, a technique involving parametric sensitivity is used to quickly identify the most uncertain input parameters; it is proposed and its efficiency is shown considering a dependability model.

- The energy consumption problem is studied also from the portable devices perspective. In particular, the case of Mobile CrowdSensing (MCS) – a contribution-based paradigm involving mobiles in pervasive application deployment and operation – is studied.

## 1.4 Structure of the thesis

The remainder of this work is divided into four parts. Part I presents basic concepts that will be considered in the rest of the thesis. Part II introduces a new framework used to optimize the Big Data applications performance. Epistemic uncertainty propagation is studied and analyzed in Part III. Finally, in Part IV the energy consumption problem is evaluated relatively to a Mobile CrowdSensing system.

More precisely, Chapter 2 discusses the main characteristics and properties of multi-class workloads, presents some of the power models available in literature and introduces the work done on epistemic (parametric) uncertainty propagation. Chapter 3 describes the model proposed to take into consideration both DVFS and multi-threading while studying the power consumption of a multi-core CPU, whereas Chapter 4 introduces the *Energy per time-unit of Execution* metric that takes into account the different nature of each job.

Chapter 5 presents the Pool depletion systems framework and some possible exploitations, in order to take advantageous scheduling decisions while considering a pool of tasks that must be executed in the shortest time.

Chapter 6 studies an M/M/1 queue under the hypothesis of uncertain inter-arrival and service rates. In Chapter 7 uncertainty propagation is applied to a power model to determine the input parameters that most affect its output measures. Chapter 8 identifies a relationship between epistemic

uncertainty and parametric sensitivity that may be used to analyze large and complex model.

Chapter 9 proposes a queuing network model to study the performance and the energy consumption of a Mobile CrowdSensing system.

Finally, Chapter 10 draws the conclusions of this work, outlining the future directions.

Several results of this thesis have been presented to international conferences. Some of them have also been published on (or accepted by) scientific journals. Finally, some new results have not been published yet.

The published contributions are:

- the power model that considers both DVFS and hyper-threading strategies of a multi-core CPU (Chapter 3) was introduced in [23] and further extended in this thesis;
- the *Energy per time-unit of Execution* metric (Chapter 4) that takes into account the class of each job has been originally proposed in [22];
- the Pool depletion systems framework (Chapter 5) has been the subject of multiple studies: in [24] a first analysis is proposed, analytical equations for the one subsystem case are given in [25] and they are extended to handle multiple subsystems in [112];
- the problem of epistemic uncertainty propagation in power models (Chapter 7) was first introduced in [63];
- the relationship between epistemic uncertainty and parametric sensitivity (Chapter 8) was investigated in [113];
- finally, the queuing network to study MCS applications (Chapter 9) has been first proposed in [111] and extended in [110].

The unpublished contributions are:

- the state of the art (Chapter 2);
- case study and numerical results of *Energy per time-unit of Execution* metric in Section 4.4;
- the experimental analysis of Pool depletion systems presented in Section 5.4.3, and the battery-operated Pool depletion systems (Section 5.5.2);
- epistemic uncertainty propagation in M/M/1 queue (Chapter 6), that has been submitted to ACM TOMPECS[1].

[1] `https://tompecs.acm.org/`

# Part I

# Preliminary work: state of the art and recurring concepts

CHAPTER 2

# State of the art

Queuing networks (QNs) are used in this work to model data-centers and study their energy consumption. In particular, the benefits of working in an optimal operational point are analyzed from the power consumption point of view. Moreover, epistemic (parametric) uncertainty propagation is also considered and its effects on the output measures of both M/M/1 queues and power models are studied.

In this chapter, available work on queuing networks, data-centers energy consumption and epistemic uncertainty propagation are analyzed and discussed.

## 2.1 Multi-class and multi-resource queuing networks

Computer systems may be modeled through the QN framework; it consists of a network of queues, where each different queue represents a service center [84]. The resources of the network are visited by the jobs (or users) that ask to be served. Once a job has been served by a resource, it is routed to another resource. A QN may be either open or closed: in an open QN the jobs arrive with a given rate (i.e., inter-arrival rate) at the model and leave once they have been completely processed; a QN is closed when the

number of customers into the model is known and does not change during the analysis. QN models are a common tool to study computer systems thanks to the low costs of definition, parameterization and evaluation, and the high accuracy of the results provided.

QN are a subset of queuing theory [84]; indeed, while queuing theory is mainly focused on modeling complex systems with a single service center and complex characteristics, QN aims to model the system using multiple service centers with simple characteristics. Moreover, several extensions of QN have been proposed; Kelly networks [78] are the most important for what concern this thesis. Differently from Jackson networks [75] that take into account only a class of customers, Kelly networks can model several classes of jobs. Although systems with single-class workload are easier to be analyzed, multi-class networks are more common since they allow to model real systems and provide more reliable results. Indeed, multi-class workloads let the modeler distinguish between requests with different behaviors, such as a system that processes both jobs that saturate the CPU and another type of jobs that saturate the I/O.

For the purpose of this thesis, multi-resource and multi-class QN are used to model the systems considered. In particular, in order to prevent the model from becoming too complex to be analyzed, but still considering reasonable approximation of the real systems, two-resource and two-class QN are used. Moreover, this kind of systems has been proven to have an optimal operational point [118] and closed-form expressions have been provided to identify it. Further details about this property are given in Section 2.1.1.

Networks of queues may be evaluated through different approaches [84]: *i)* performance bounds analysis, a simple technique that allows evaluation with very low complexity; *ii)* computation of the values of performance metrics, obtained with complex but efficient algorithms; *iii)* QN simulation, that is more flexible but also more expensive than other approaches. The techniques that have been mainly adopted in this thesis are the analytical and simulative ones.

### 2.1.1 Common saturation sector and equi-utilization point

Closed QN with two resources and two-class workloads have been deeply analyzed by Rosti et al. in [118]. In particular, they proved that an optimal operational point – where the system power[2] [80] and the sum of the utilization of all the resources are maximized – exists. They provided

[2]It is defined as the ratio of system throughput to system response time, thus it increases when the system throughput increases or the system response time decreases.

closed-form expressions to derive such a point, just assuming to know the matrix $[D_{rc}]$, where $D_{rc}$ is the service demand of a class $c$ job when it is served by a resource $r$ (i.e., the amount of service time required to resource $r$ by a complete execution of a class $c$ job).

The population mix of a QN with multi-class workload is defined as a vector of fraction of customers of each class, $\vec{\beta}$. Indeed, if $N_A$ and $N_B$ are the number of customers in the system for class $A$ and class $B$, respectively, the population mix is derived as:

$$\vec{\beta} = (\beta_A, \beta_B) = \left( \frac{N_A}{N}, \frac{N_B}{N} \right) \tag{2.1}$$

where $N = N_A + N_B$. The optimal operational point coincides with the equi-utilization one. Since a two-resource model is considered, in this point it is $U_1 = U_2$, where $U_r$ is the global utilization of resource $r$.

In order to determine the equi-utilization point, also known as optimal population mix, $\vec{\beta}^*$, the following equation is given in [118]:

$$\vec{\beta}^* = \left( \beta_A^* = \frac{ln\frac{D_{2B}}{D_{1B}}}{ln\frac{D_{1A}D_{2B}}{D_{1B}D_{2A}}}, \beta_B^* = 1 - \beta_A^* \right) \tag{2.2}$$

Note that, the equi-utilization point only depends on the service demands, $D_{rc}$, and not on either the number of class A and class B jobs or their sum, $N$.

Besides the equi-utilization point $\vec{\beta}^*$, also the equi-load point may be derived. Although two equally loaded stations may be expected to be also equally utilized, it is not true. Indeed, the equi-load point, $\vec{\alpha}^*$ is computed as:

$$\vec{\alpha}^* = \left( \alpha_A^* = \frac{D_{2B}D_{1B}}{D_{1A} + D_{2B} - D_{1B} - D_{2A}}, \alpha_B^* = 1 - \alpha_A^* \right) \tag{2.3}$$

thus resulting in a different point with respect to the equi-utilization one.

Finally, the equi-utilization point is shown to belong to the common saturation sector [5,118] (see Figure 2.1). It is defined as a sector of population mixes for which both the resources saturate and become the bottlenecks of the system. For example, in the case of $D_{1A} > D_{2A}$ and $D_{2B} > D_{1B}$ the bottleneck of the system is resource $1$ when $\vec{\beta} = (1, 0)$ (i.e., only class A jobs are into the system) and resource $2$ when $\vec{\beta} = (0, 1)$ – being the bottleneck of a single class model defined as the resource with the largest service demand [40]. For this reason, there is at least a population mix for which the bottleneck of the system switches from resource $1$ to resource $2$.

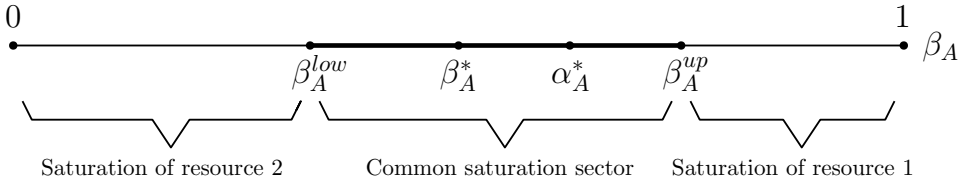



**Figure 2.1:** *Representation of the common saturation sector (bold line), equi-utilization and equi-load points defined in [118].*

Starting from this consideration, the edges (i.e., population mixes) of the common saturation sector, $\vec{\beta}^{low}$ and $\vec{\beta}^{up}$, are defined as follows:

$$\begin{aligned}\vec{\beta}^{low} &= \left(\beta_A^{low} = D_{2A}\frac{D_{2B} - D_{1B}}{D_{1A}D_{2B} - D_{2A}D_{1B}}, \beta_B^{low} = 1 - \beta_A^{low}\right)\\ \vec{\beta}^{up} &= \left(\beta_A^{up} = D_{1A}\frac{D_{2B} - D_{1B}}{D_{1A}D_{2B} - D_{2A}D_{1B}}, \beta_B^{up} = 1 - \beta_A^{up}\right)\end{aligned} \tag{2.4}$$

When the population mix is outside the common saturation sector, only one of the two resources is saturating, otherwise both of them are the bottlenecks of the system. Similar results may be obtained for the case $D_{2A} > D_{1A}$ and $D_{1B} > D_{2B}$.

Figure 2.1 depicts all the population mixes with which a system may work, that are all the values between $\beta_A = 0$ (i.e., $\vec{\beta} = (0, 1)$) and $\beta_A = 1$ (i.e., $\vec{\beta} = (1, 0)$). Considering the case $D_{1A} > D_{2A}$ and $D_{2B} > D_{1B}$, resource 2 is the bottleneck of the system when $0 \leq \beta_A < \beta_A^{low}$. On the contrary, resource 1 saturates when $\beta_A^{up} < \beta_A \leq 1$. Finally, both the resources are saturating when the system works with a population mix such that $\beta_A^{low} \leq \beta_A \leq \beta_A^{up}$ (i.e., the population mix is in the common saturation sector). In the same figure, it is also shown that the equi-utilization point, $\beta_A^*$, and the equi-load one, $\alpha_A^*$, belong to the common saturation sector, but are distinct.

## 2.2 Power and energy models and strategies

The ever-increasing data-centers energy consumption leaded computer scientists to develop techniques to decrease the amount of energy a data-center needs to properly work.

Although power consumption models and energy-saving techniques have been proposed for many of the aspects that affect the data-centers performance (e.g, computer architecture, operating system, algorithms), for the purpose of this work only performance metrics (e.g., throughput, response

time, resources' utilization, etc.) have been considered, thus providing a better view on architectural aspects.

### 2.2.1 Basic concepts

As stated in [10], it is fundamental to have a deep knowledge of the basic concepts related to power and energy consumption in order to better understand the mechanisms used for their management. The power of a system is the rate at which the work is performed, it is measured in watts (W) and it is one watt when one Ampere is transferred through a potential difference of one volt. Instead, energy is defined as the amount of work done in a period of time and is measured in Joule (J) or watt-hour (Wh), with $1\ Wh = 3600\ J$. Energy equivalent to a watt-hour is consumed when the power of one watt is run for one hour.

Power and energy are computed through Eqs. (2.5) and (2.6), respectively:

$$P = V \cdot I \tag{2.5}$$

$$E = P \cdot T \tag{2.6}$$

where $V$ is the voltage (measured in volts), $I$ is the electric current (measured in Amperes) and $T$ is the period of time considered.

For the purpose of this work, the difference between power and energy is essential; indeed, reducing power consumption does not implies that also energy consumption is decreased [10]. First of all, consider the distribution of costs between power and energy. Power inefficiency is mainly due to underutilized resources – although the resources are used for less than $50\%$ on average, the infrastructure must be built to manage the peak load that rarely occurs – thus, it is an infrastructure cost. Instead, energy consumption accounts for electricity bills, which is the main cost of a data-center. Then, let us assume the CPU power consumption (i.e., its performance) is reduced to the purpose of reducing the energy consumed by the CPU itself. If it is done, the time required by the CPU to complete a given task is stretched. Since energy is computed as shown in Eq. (2.6), the energy consumed by the underpowered CPU may be as high as the one used by a more powerful CPU that stops working after a shorter time.

### 2.2.2 Energy-aware techniques classification

Several authors recognize the existence of a correlation between the utilization of the resources and their energy consumption [9, 12, 79]. In fact, even if a resource is idle (i.e., its utilization is $U = 0\%$), some power is required

to keep that resource on. As stated in [49], the amount of power required to keep on a server is not lower than half of the peak power (i.e., the power needed when the server is fully utilized). For this reason, turning off the under utilized servers may help to decrease a data-center energy consumption. Such a decision must be taken considering also the service level agreements (SLAs) between the service provider and the users, since providing reliable quality of service (QoS) is an important requirements for data-centers and cloud computing [10]. Indeed, even if a shut down server does not consume energy, it may require several minutes to be turned on when needed, thus slowing down the new processing, increasing the response time and decreasing the throughput. Also reducing the number of available servers may degrade the performance due to an increased load on the remaining ones. For this reason, the trade-off between utilization of the resources and time required to execute a job is a crucial factor which must be considered when trying to minimize the energy consumption of a data-center.

Several approaches have been proposed in order to cope with the energy consumption problem in data-centers and they have been classified by Jin et al. [76]. They identified four main categories to describe the techniques used to improve the data-centers energy efficiency, which are represented in Figure 2.2.

**Energy efficiency techniques**. It is the main approach applied to decrease the energy consumption of computing devices. Tools and frameworks are exploited to this purpose. Dynamic voltage/frequency scaling (DVFS) is a widely adopted technique belonging to this set of strategies. It focuses on reducing chip's voltage and frequency in order to decrease its energy consumptions. Besides decreasing its speed, a device may be also turned off through power-down mechanisms. Indeed, when a device is idle (i.e., it is on but is not used), it may be shut down in order to save energy. Other technique are used to increase the CPU utilization (thus, its power consumption); it is the case of simultaneous multi-threading (SMT) that enables the parallel execution of multiple independent threads.

**Resource management**. Virtualization and cloud computing are also used to increase the energy efficiency of a data-center. As stated in [10], while virtualization allow to create several virtual machines (VMs) on a physical server, thus increasing its utilization, cloud computing is a power efficient framework since it allows to adjust resource usage depending on users' current requirements and the cloud provider may efficiently arrange all the incoming requests. In fact, the main technique belonging to this category is the workload consolidation that lets the provider increase the efficiency of the data-center, for example increasing the resources utiliza-

tion, thus decreasing the global execution time. Other techniques are the network traffic engineering, i.e., frameworks that allow to aggregate and redirect traffic to make some resources enter a low-power state, the power distribution that allows to better allocate the available power, and the use of renewable energy.

**Green metrics**. Cloud providers need appropriate metrics to evaluate the greenness of their data-centers and the efficiency of the energy saving techniques adopted. The parameters such as the carbon dioxide emissions and the efficiency of the data-centers should be clearly visualized by the cloud providers in order to take suitable decisions about how to increase their revenues. Moreover, monitoring, controlled experiments and simulations may provide crucial information during the data-center design phase.

**Thermal control**. High power consumption and the increasing density of data-centers' components make heat dissipation grow. This is a data-center critical problem for several reasons; since the reliability of each component is affected by the data-center temperature, cooling mechanisms are required, thus power consumption further rises. It has been estimated that, for each watt consumed by a resource, $0.5$ to $1$ more watt is required for the cooling system [10]. In fact, almost half of the energy consumed by the data-centers is for the cooling [36, 49, 93]. However, recent studies [47] concluded that data-centers can operate with hotter temperature than the current ones, without excessively reduce the life of the components, thus decreasing the energy consumption.

In this thesis the first three approaches have been considered. Indeed, as previously said, a power model to consider DVFS and multi-threading is deployed, a workload consolidation problem is studied, and a metric to better evaluate energy consumption in multi-class workload systems is proposed.

### 2.2.3 Energy saving algorithms and frameworks for data-centers

Several energy saving techniques available in literature have been analyzed by [10] and [67]. According to Beloglazov et al. [10], the available algorithms and strategies may also be distinguished based on the environment considered: non-virtualized or virtualized.

Resource management, scheduling algorithms and power management are the main strategies adopted to decrease energy consumption of non-virtualized systems [28, 48, 54, 57, 114, 128]. Usually, besides a monitoring phase, where the workload (e.g., the incoming requests to a web server) is observed, the technique adopted change the current system configura-

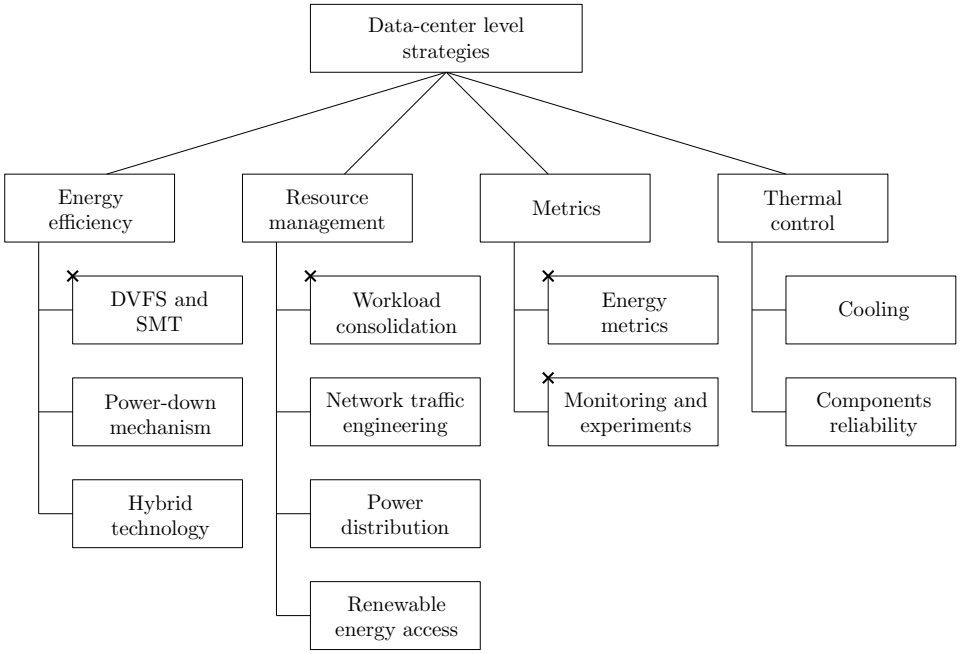



**Figure 2.2:** *The green data-centers techniques identified in [76]. The strategies adopted in this thesis are marked by × on the top-left corner.*

tion. For example, it may turn on/off the physical nodes, change the CPU frequency or schedule the requests on different resources to decrease the system energy consumption while complying with the SLAs.

Pinheiro et al. [114] proposed an algorithm to spread (i.e., load balancing) or concentrate (i.e., load unbalancing) the incoming requests on either all the nodes or just a few of them, respectively. While the balancing technique is adopted to reach required performance levels and comply with the SLAs, the load concentration is used to make some nodes idle, thus being able to shut them down and save energy. The algorithm considers three different resources (CPU, disk and network), but it is claimed to be implemented on a single node – thus, resulting in a single point of failure and possibly becoming the bottleneck of the system – and be able to turn off one node at a time, potentially resulting in bad performance when applied to large clusters.

Chase et al. [28] proposed an operating system (OS) for Internet hosting centers, that works with the OS of the available servers to coordinate the data-centers' components. Besides efficiently scheduling the incoming requests taking into account power consumption and SLAs, the OS proposed also decrease the power consumption of the CPU of each server. This is

possible by continuously monitoring the workload and allocating the resources to each application based on the QoS selected for that request. It is claimed that this technique allows a data-center to adapt to fluctuating workload while improving energy efficiency.

The algorithm proposed by Elnozahy et al. in [48] uses both VOVO (i.e., vary on vary off, or turning on/off the available physical resources) and DVFS to manage the power-performance trade-off. In particular, the authors investigated different policies to be adopted by their algorithm. They claimed the Coordinated Combined Policy is the most efficient one and it operates in three steps: *i)* it determines the appropriate CPU frequency based on the required response time; *ii)* it derives the number of servers that must be on, and shut down all the other resources; *iii)* it sets the CPU frequency of all the available nodes proportionally to the number of nodes that are on.

The strategy proposed in [128] exploits the workload consolidation in order to find an optimal utilization point, such that the energy consumption is minimized and the system performance is increased. Indeed, the energy consumption is high with low utilization due to the servers idle period; on the contrary, the system performance are affected by large resources utilization (e.g., high cache miss rate, scheduling conflicts, etc.). CPU and disk are the components considered in this work.

Gandhi et al. [54] studied the problem of finding an optimal power allocation for server farms. In particular, they considered the power consumption being affected by the CPU and its frequency. After noting a linear power-to-frequency relationship for the DVFS technique, Gandhi et al. tried to identify the optimal operating point such that the server power consumption and the average response time are minimized.

Garg et al. [57] studied the problem of energy efficiency executing high performance computing (HPC) applications from a cloud provider perspective. They proposed some scheduling strategies that exploit DVFS and the data-centers geographical distribution. Some of them account only for either the provider profit or the $CO_2$ emissions. The authors also proposed a multi-objective policy that takes into considerations both the profit and the environmental aspects. After finding for each application the data-center with the lowest carbon dioxide emissions, the multi-objective scheduling policy selects the application/data-center pair that maximizes the provider's profit.

Also the case with virtualized environment has been largely investigated in literature [16,83,105,127,139]. The strategies proposed usually take into account the CPU power consumption and work with DVFS and VM con-

solidation techniques in order to decrease energy consumption and satisfy the performance requirements.

In [105] the authors used a *soft resource scaling* in order to allow the virtual machine monitor (VMM) to limit the time a virtual machine can use the resource. Furthermore, the authors found that greater benefits may be obtained using both hard and soft scaling. Thus, they proposed local and global policies for their scaling technique: the local policy is responsible for the coordination of the virtual machines' energy-saving techniques; the global one manages the physical machines.

Kusic et al. [83] took into consideration the case of dynamic VM allocation for a multi-tiered web application that executes a variable workload. They adopted a limited lookahead control in order to maximize the service provider's profit, and took into consideration different parameters: *i)* the number of VMs to be allocated to each request; *ii)* the number of host on which each VM should be placed; *iii)* the amount of CPU shared for each VM; *iv)* the number of physical machines that may be turned off. Moreover, a Kalman filter is adopted to estimate how the workload is changing, thus to forecast the future system requirements and resources allocation.

The authors of [127] adopted a strategy to decrease the energy consumption, while keep on complying with the QoS. In particular, the resources are allocated to each application's VM based on the priority of the application itself. This strategy is thought for enterprises, where the priority of each application may be exactly known. Three schedulers are used to reach their goal: the application-level scheduler assign each incoming request to an application's VM; the local-level scheduler allocates the available resources to each VM based on the application's priority; the global-level scheduler controls the resources flowing among the applications. They do not consider VMs migration to improve the system adaptation to dynamic workloads.

*pMapper* is a framework proposed by Verma et al. in [139]. It implements three different managers to dynamic place applications in a virtualized environment, minimize the system energy consumption and comply with the required QoS. In particular, the *performance manager* is in charge of monitoring each application and resizing their VMs according to the SLA. The *power manager* changes the CPU's power state and applies DVFS. The *migration manager* deals with VM migration in order to consolidate the workload. The three managers work at each time frame in order to continuously optimize the service provider's profit.

Boru et al. [16] proposed a data replication technique in order to better exploit the data-centers geographical distribution to improve energy con-

sumption and network bandwidth, and decrease the communication delay. After modeling the three aspects to improve, they proposed an algorithm that, starting from the original data object available in the *central DB*, creates and distributes its replicas on *data-center DB* and *rack DB* based on the collected observations of frequency of accesses.

### 2.2.4 Power and energy consumption models

Several power models have been proposed for studying and developing new techniques to decrease data-centers power consumption. The linear model proposed by Fan et al. in [49] has been widely used in literature [18, 50, 151]. In this model, the server power consumption has been estimated to linearly grow with CPU utilization:

$$P(U_{cpu}) = P_{idle} + (P_{busy} - P_{idle}) \cdot U_{cpu} \tag{2.7}$$

where $P_{idle}$ and $P_{busy}$ are the power consumptions of an idle and a fully utilized servers, respectively, whereas $U_{cpu}$ is the CPU utilization. This model is adopted in [151] beside a cooling power model to identify the data-center cooling energy saving potential. It is also used in [50] to predict the racks performance and the thermal dynamics in order to provide a predictive control able to save a significant amount of energy just taking decisions for server provisioning, job placement and thermal management. Fan et al. also introduced a non-linear model where power consumption increases monotonically with the CPU utilization. The equation that defines the non-linear model is:

$$P(U_{cpu}) = P_{idle} + (P_{busy} - P_{idle}) \cdot (2U_{cpu} - U_{cpu}^{r}) \tag{2.8}$$

where $r$ is a calibration parameter that is estimated through experiments.

Buyya et al. [18] proposed a power model similar to the one introduced in [49] in order to deal with the energy-aware resource allocation in cloud environments. Since the ratio between $P_{idle}$ and $P_{busy}$ is generally known (e.g., $70\%$), they adopted the model:

$$P(U_{cpu}) = c \cdot P_{busy} + (1 - c) \cdot P_{busy} \cdot U_{cpu} \tag{2.9}$$

where $c = P_{idle}/P_{busy}$. This power model has also been used in [21] to propose an adaptation strategy for managed Cassandra data centers that can minimize their operational costs while complying with the SLAs.

Those are the simplest available power models due to the small number of required parameters. Nonetheless, several other more accurate power models have been proposed in literature. Note that, the cost of a greater

accuracy is the complexity of the model itself. For example, some power models consider several server components besides the CPU. It is the case of the linear weighted power model proposed by Economou et al. [45], that is:

$$P = P_{idle} + U_{cpu} \cdot l_{cpu} + U_{mem} \cdot l_{mem} + U_{disk} \cdot l_{disk} + U_{net} \cdot l_{net} \quad (2.10)$$

where $U_r$ is the utilization of resource $r$, and $l_r$ represents the linear relationship between the utilization and power consumption for each resource $r$. In this case, a weighted linear model is used to estimate the power consumption of a server, that is the sum of power consumption of each component (i.e., CPU, memory, disk and network).

A similar approach is adopted by Bohra et al. in [14]. They first proposed a power model that takes into consideration CPU, cache, memory and disk, then – after noting a correlation between CPU and cache, and memory and disk – improved it as follows:

$$\begin{aligned} P_{cpu,cache} &= P_{idle,1} + c_{cpu} \cdot \xi_{cpu} + c_{cache} \cdot \xi_{cache} \\ P_{mem,disk} &= P_{idle,2} + c_{mem} \cdot \xi_{mem} + c_{disk} \cdot \xi_{disk} \\ P &= c_1 P_{cpu,cache} + c_2 P_{mem,disk} \end{aligned} \quad (2.11)$$

where $P_{idle,1}$ and $P_{idle,2}$ are the idle power consumptions, $c_r$ is the weight of the monitored system events $\xi_r$ for resource $r$, and $c_1$ and $c_2$ are the contribution ratios of the power consumptions $P_{cpu,cache}$ and $P_{mem,disk}$, respectively.

Other available models take into consideration the power consumption of the whole data-center. For example, Gosh et al. [59] proposed a stochastic model to evaluate the performability of an IaaS Cloud environment and its cost. In particular, they grouped all the physical machines into three different pools (i.e., hot, warm and cold) based on their current status (i.e., running, turned on and not ready, turned off, respectively). Thus, they gave different costs to performance and dependability parameters in order to evaluate and eventually optimize the cost and capacity of the considered environment. While considering the power consumption of the physical machines and its cost, they also take into account the cooling cost.

All the models presented in this Section are meant to estimate the power consumption of a component (e.g., the CPU), a server or the whole data-center. For the purpose of this thesis, it is not always enough to take into consideration the power required by the system considered, but it is necessary to know also its energy consumption. The energy models may be derived starting from all the power models presented in this Section (or other

power models available in literature), and applying Eq. (2.6) to estimate the energy consumption.

Finally, also stochastic hybrid models worth to be mentioned, although we have not recurred to them in this thesis. They combine discrete-event and continuous simulation techniques to efficiently represent both the components with a discrete behavior and those that follow a continuous evolution. In fact, when applied to energy consumption related problems, stochastic hybrid models have been adopted for many purposes, e.g., modeling energy management on smartphones [1], or studying energy storage in smart homes while considering how it is affected by grid-convenient battery management policies [73].

## 2.3 Uncertainty propagation

Several frameworks have been proposed to classify uncertainty [119] and the two main categories that have been identified are *aleatory* and *epistemic uncertainty* [108]. The former, also known as irreducible uncertainty, depends on natural variations of the physical phenomena and represents randomness in samples; an improved model is required in order to decrease this type of uncertainty. Instead, epistemic uncertainty is due to a lack of knowledge about the physical systems. It is the kind of uncertainty considered in this work since it may be reduced [4].

Although several papers in literature have already considered QN models with uncertainty [17, 58, 117], they took into account only aleatory uncertainty.

For example, Boxma and Kurkova [17] studied the workload tail behavior in the case of an M/M/1 queue with different service rates, that are exponentially distributed high-speed periods and low-speed periods with a changing distribution.

Both inter-arrival and service rates of an $M/G/1$ queue are assumed to vary over time in [58]. The model proposed by Gelenbe and Roseberg is used to analyze those systems in which the parameters may vary slowly, such as telephone systems or computer networks with bursty workloads. The authors set the *slow changes assumption* in order to make each variation of the input parameters lead to some exploitable approximations.

Rosenberg et al. [117] took into consideration random variations in the inter-arrival rate of exponential queuing systems. In particular, they first derived the necessary and sufficient conditions for queue stability when the buffer size is infinite, then focused on finite buffer with the incoming requests that stop entering the queue when the buffer is full.

In all the previous cases, expected value of the model's parameters may change at any time due to random environments. Since the input parameters are assumed to vary during the analysis of the model, the problem analyzed in those papers is different from the kind of uncertainty considered in this thesis, where the average value of the input parameters is assumed to be constant.

Some applications of epistemic (parametric) uncertainty is studied in [88, 131]. In particular, authors of [131] proposed a numerical method that is applied to an $M/G/1/K$ queue with vacation when only one input parameter is affected by epistemic uncertainty (i.e., the vacation rate). Moreover, they assumed the distribution of the uncertain parameter is given. Lopatatzidis et al. [88] used parametric uncertainty to study a $Geo/Geo/1/K$ queue. They considered a finite state system and assumed bounds on the input parameter values instead of the distributions.

Although the same kind of uncertainty is considered in this thesis, it is different from [88, 131] for several reasons. In particular: *i)* at least two input parameters are assumed to be affected by epistemic uncertainty; *ii)* the M/M/1 queue stability condition is taken into account since inter-arrival and service rates are assumed to be the uncertain input parameters; *iii)* a given confidence interval on the input parameters is used to derive epistemic distributions; *iv)* both closed form expressions and a numerical approach are adopted to propagate the uncertainty through the model.

Recently, Bortolussi et al. [15] analyzed uncertain stochastic models by introducing a new class of models, namely imprecise population processes, which captures the parameters variability over time and use differential inclusions to derive the evolution of their probability mass. This approach is based on [126], where Markov chains have been extended with interval probabilities, i.e., transition probabilities are not precisely known but regulated by the probabilities associated to the set of all elementary sets within lower and upper bounds. However, both approaches deal with uncertain parameters that are not regulated by distribution functions supporting the deduction of closed form formulas, whereas, in this thesis, such formulas are used to calculate the performance metrics (i.e., average number of entities in the system and average response time).

Epistemic uncertainty propagation has been mostly studied for dependability models adopting different techniques [7, 97, 98]. In [98] parametric epistemic uncertainty propagation through analytic dependability models is presented. Closed-form expressions for the distribution function, expected value and variance of model outputs are derived for calculating the system reliability, and the model output results to be a simple closed-form expres-

sion of model input parameters. Authors of [98] used multidimensional integration proposed by Singpurwalla in [125] to analyze the uncertainty propagation; the same technique is also used in this thesis when studying uncertainty propagation in M/M/1 queues and power models. Baumgartel et al. [7] adopted a simulative technique. In particular, they used inverse uncertainty propagation on input parameters in order to find the input uncertainties that satisfy a given bound on the simulation output uncertainty. Also note that, when dealing with complex and large systems, it becomes hard to threat with multidimensional integration. For this reason, alternative techniques to multidimensional integration have been proposed in literature, such as the Monte Carlo sampling method to propagate epistemic uncertainty in complex and large models proposed and adopted by Mishra et al. in [97].

The Monte Carlo sampling method is also used for calculating: *i)* reliability in [92], where a diverse set of parameter range distributions self-regulate the number of architectural evaluations to the desired significance level and report the desired percentiles which ultimately characterize the system reliability; *ii)* performance in [134], where parameters uncertainties are sampled from probability distribution functions and propagated in multiple software architectural models, thus to evaluate their robustness with respect to a certain set of performance requirements.

The specification of uncertain parameters in literature leads to classify the proposed approaches into two categories: *i)* parametric or probabilistic, i.e., the probability of density functions (pdf) is known for uncertain parameters, and several solving techniques are available to deal with such specification, e.g., maximum likelihood [39], or applied statistics such as method of moments and Bayesian estimation [102]; *ii)* non-parametric or fuzzy (originally introduced by Zadeh [147]), i.e., uncertain parameters are described using linguistic categories with fuzzy boundaries.

Looking at the general context of non-functional analysis of software under uncertainty, several approaches are based on sensitivity analysis techniques that aim to identify the model parameter ranges affecting the software quality, e.g., [120]. In the software performance domain, one of the seminal works in this direction is [43], where the concepts of uncertainty, performance conditions and implications firstly appeared.

### 2.3.1 Multi-dimensional integral for epistemic uncertainty propagation

In this thesis, when dealing with epistemic uncertainty propagation, the multi-dimensional integral approach, adopted in [98, 125], is used. In particular, consider a set of input random variables $\{\Theta_i, i = 1, 2, ..., l\}$ and an output measure $M$ defined as a random variable $M = g(\Theta_1, \Theta_2, ..., \Theta_l)$. Computing the output measure with specific parameter values can be seen as computing the conditional measure $M(\Theta_1 = \theta_1, \Theta_2 = \theta_2, ..., \Theta_l = \theta_l)$ (henceforward denoted by $M(\bullet)$) because of the uncertainty introduced by the input parameters. Applying the theorem of total probability [133] and using the joint epistemic density $f_{\Theta_1,\Theta_2,...,\Theta_l}(\theta_1, \theta_2, ..., \theta_l)$ (hereinafter denoted by $f(\bullet)$), the conditional measure $M(\bullet)$ can be unconditioned. Thus, Cdf of the measure $M$ is computed as follows:

$$F_M(m) = \int ... \int \mathbb{1}(M(\bullet) < m) \cdot f(\bullet)\, d\theta_1 \, ... \, d\theta_l \tag{2.12}$$

where $\mathbb{1}(\zeta)$ is an indicator variable that is $1$ when the event $\zeta$ is verified, and $0$ otherwise. Instead, the expected values of measure $M$ is computed as:

$$\mathbb{E}[M] = \int ... \int M(\bullet) \cdot f(\bullet)\, d\theta_1 \, ... \, d\theta_l \tag{2.13}$$

It is also possible to derive the variance of the measure $M$ as $Var[M] = E[M^2] - E[M]^2$, where the second moment is computed as:

$$\mathbb{E}[M^2] = \int ... \int M^2(\bullet) \cdot f(\bullet)\, d\theta_1 \, ... \, d\theta_l \tag{2.14}$$

For the sake of simplicity, in this thesis, we assume the epistemic random variables to be independent, thus the joint epistemic density is given by the product of the marginals:

$$f(\bullet) = f_{\Theta_1,\Theta_2,...,\Theta_l}(\theta_1, \theta_2, ..., \theta_l) = f_{\Theta_1}(\theta_1) \cdot f_{\Theta_2}(\theta_2) \cdot ... \cdot f_{\Theta_l}(\theta_l)$$

CHAPTER 3

# CPU power model for DVFS and simultaneous multi-threading

## 3.1 Motivation

The servers' CPUs are the components that are usually taken into account while estimating the power consumption of a data-center. As said in Chapter 2, it has been proven [49, 138] that server power consumption and its CPU utilization are strongly related and the former metric depends on the latter one; consequently, CPU utilization also affects the energy efficiency of the whole data-center.

Dynamic voltage/frequency scaling (DVFS) and Simultaneous multi-threading (SMT) are two technologies available in literature and used to tackle the energy efficiency problem. DVFS has been developed to make CPU voltage manageable; indeed, it may be increased if it is required to comply with the SLAs, or decreased to reduce server's power consumption and save energy. Instead, SMT was first introduced for performance reasons, since it allows the CPU to increase its utilization and throughput. However, it has also been crucial while considering the power consumption issue, since it gives control over the CPU utilization. In fact, it is possible to reduce the CPU idle periods by enabling SMT, thus decreasing the servers'

energy consumption.

Although DVFS and SMT are two very common techniques used to improve data-centers energy efficiency, some recent power models and frameworks do not account for them. In particular, they are rarely considered in the same model and at the same time, thus decreasing the accuracy of the framework.

In this chapter the linear power model proposed by Fan et al. in [49] – that has been widely adopted in literature to estimate the servers' power consumption – is extended to make it account for DVFS and SMT at the same time and provide more accurate power consumption estimates.

## 3.2 What are DVFS and SMT?

In this Section the basic definitions of dynamic voltage/frequency scaling and simultaneous multi-threading are given, and the papers in which these two techniques are adopted to decrease a server energy consumption are discussed.

### 3.2.1 Dynamic Voltage and Frequency Scaling

As said in Section 2.2.3, DVFS is commonly described in literature as one of the main available energy saving techniques. It was firstly adopted in [141] in order to decrease the power consumption of a CMOS integrated circuit, such as the CPU component. In fact, the CPU power consumption is proportional to $V^2 \cdot f$ [29] and may be computed as [86]:

$$P = P_{idle} + C \cdot V^2 \cdot f \tag{3.1}$$

where $C$ is the capacitance of the transistors, $f$ is the operating frequency and $V$ is the voltage. Since the commonly given relationship between voltage and frequency is $V \propto f$ [29], it possible to decrease the number of CPU cycles per second in order to reduce the voltage and save energy.

DVFS is used to dynamically adapt the CPU power to the incoming requests; it is usually enabled accordingly to the system workload that may be either predicted at run time or given in advance [30]. In the former case, based on the predicted incoming requests, the resources (e.g., the CPU) may increase or decrease their frequency (and their voltage) on-line in order to process all the tasks with the minimum energy consumption, but still complying with the SLAs. Instead, the latter case allows the service provider to set the frequency of the resources a priori based on the available knowledge about they system. For instance, if the memory is the bottleneck of the system considered, the service provider may decrease the frequency

of the CPU in order to save energy without affecting the performance of the system.

Accordingly to the previous considerations, Choi et al. [30] proposed a *workload decomposition* technique in order to distinguish between on-chip and off-chip requests: on-chip requests are executed on CPU, whereas off-chip requests are served by memory (the second resource considered). Exploiting the data cache misses and the CPU stall cycles, the authors proposed a DVFS policy that dynamically change the CPU frequency in order to maximize the energy-performance trade-off.

DVFS technique is also applied to components other than CPU. Indeed, David et al. [35] proposed to use DVFS technique for memory in order to dynamically adapt its operating point to the current workload, after noting that a large amount of memory power depends on its frequency. In fact, they used an algorithm to set the memory frequency/voltage based on its bandwidth utilization.

### 3.2.2 Simultaneous Multi-Threading

SMT is a technique used to improve the efficiency of threads parallel execution, in particular enabling independent threads to issue multiple instructions in a single cycle [135]. The main SMT impacts are higher CPU utilization and throughput [87, 123, 135], since the same number of requests can be served in a shorter time when SMT is enabled. The CPU power consumption also increases due to the relationship between CPU utilization and its power consumption, already discussed in Chapter 2. However, since the execution time is shorter, its energy consumption may be reduced.

Intel Hyper-Threading Technology (HTT) is the Intel's implementation of SMT. It was first available on Xeon server processors in February 2002 and nowadays it is a widespread technology. In particular, each available CPU core is seen by the OS as two logical cores that share the incoming requests when feasible.

SMT has been largely taken into consideration in literature [87,123,135, 149] due to its importance.

Tullsen et al. [135] used simulation to evaluate the improvements made available by SMT. In particular, they showed the SMT technology is able to outperform other parallel enabling strategies (e.g., single-threaded wide superscalar, fine-grained multi-threading, multiple-issue multiprocessor) improving processors' utilization and throughput, but also increasing the complexity of instructions and the shared resources contentions.

Seng et al. [123] proved that SMT also increases the CPU's power con-

sumption for the same reasons that increase processor's performance. Indeed, SMT can decrease energy consumption providing more parallelism and minimizing the underutilized resources. For that purpose, they used a simulative technique and proposed a power model that splits the CPU considered into several different areas, and all of them concur to the total CPU power consumption.

Li et al. [87] first extended an available simulator to make it account for SMT technology, then investigated how SMT affects the micro-processor performance and its power consumption, i.e., the SMT power-performance trade-off. They found the SMT can increase the performance gain and power consumption by $20\%$ and $24\%$, respectively, and significantly reducing the energy consumption.

Zhai et al. [149] took into consideration the Intel Hyper-Threading Technology and proposed *HaPPy*, a power profiling to dynamically estimate the threads power consumption. In other words, *HaPPy* can split the server power consumption among all the threads processed by the server itself, thus estimate the power consumption of all the application concurrently executed.

## 3.3 Testbed and experiments for DVFS and SMT performance evaluation

In this Section the test environment and the experiments used to evaluate how DVFS and SMT affect the CPU power consumption are described. Problem analysis and data collection are required in order to propose a new power model able to accounts for the two energy saving techniques.

### 3.3.1 Testbed description

In order to collect different results and provide a more accurate power model, data have been collected from two hardware configurations:

- *Machine1* is an HP computer with an Intel i3-2120 CPU@3.3GHz, two CPU cores and a 6GB memory (4GB + 2GB), a hard disk drive and an integrated graphic card, with Ubuntu 12.04 OS. Its idle and maximum power consumption have been derived from measurements and are $P_{idle} = 29$ watt and $P_{busy} = 66$ watt, respectively.
- *Machine2* is an ASUS server with an Intel i7-3770 CPU@3.4GHz, with four cores, a 16GB memory, a dedicated GeForce GTX 560 GPU, two hard disk drives and a solid state disk. For this configuration, Ubuntu 14.04 is the running operating system. The idle and maximum

power consumption have been estimated to be $P_{idle} = 59$ watt and $P_{busy} = 115$ watt, respectively.

Both the CPU micro-processor support Intel HTT technology and the SMT level is two for both the configurations; that means that Machine1 and Machine2 can concurrently run four and eight parallel threads, respectively, with a dedicated logical core and without needing of threads interleaving. Moreover, the two operating systems used for these experiments enable the dynamic voltage/frequency scaling, thus allowing to govern the CPU frequency based on the current workload.

In particular, it is possible to manually set the maximum CPU frequency to a specific value updating the `scaling_max_freq` file, and turning on/off each logical thread by editing the `online` file. Thus, it is allowed to turn off HTT technology for all the processor or just for some cores.

Both the machines are connected to the power source via a *Yokogawa WT210 digital power meter*[3] in order to periodically measure their power consumption. The CPU statistics, including its utilization, are monitored with the *iostat*[4] command that derives each measure as the mean of all the threads that are turned on. Finally, data about frequency of each thread may be found periodically reading the `/proc/cpu_info` file.

Since the goal of these experiments is to investigate the CPU power consumption with DVFS and SMT technologies, a benchmark that executes CPU-bound applications is required. In particular, each experiment is executed generating instances of the *Sunflow* benchmark from the *DaCapo* suite [13]. *Sunflow* renders a set of images using ray tracing and splitting the load into several concurrent threads.

### 3.3.2 Experiments analysis

As said, DVFS and SMT have been proven to deeply affect CPU performance and server's power consumption and energy efficiency. Indeed, DVFS can change the clock frequency (and the voltage) of the processor, while SMT can increase its utilization.

However, their impact on server's energy consumption is hard to detect when considering only the CPU utilization. For this reason, we perform several experiments in order to investigate how the dynamic voltage/frequency scaling and simultaneous multi-threading affect the CPU performance.

[3] `http://tmi.yokogawa.com/discontinued-products/digital-power-analyzers/digital-power-analyzers/wt210wt230-digital-power-meters/`. Accessed: Jan. 15, 2018.

[4] `https://linux.die.net/man/1/iostat`. Accessed: Jan. 15, 2018.

In particular, we collected data from *Machine1* and *Machine2* while executing *Sunflow* benchmark and considering different configurations. In fact, the two machines have been analyzed setting the maximum CPU frequency and the number of available logical threads to different values for each experiment, as shown in Section 3.3.1.

Concerning *Machine1*, the maximum CPU frequency was set to $\{1600, 2500, 3300\}$ MHz. Note that, when the maximum frequency is different from the minimum one, CPU may work with every value of frequency that is between the minimum and the maximum one. Thus, for example, by selecting $1600$ MHz as maximum frequency (i.e., $\text{freq}_{\text{max}} = \text{freq}_{\text{min}}$) the frequency scaling is disabled, whereas if the maximum frequency is $2500$ MHz the CPU speed may span from $1600$ to $2500$ MHz. The number of concurrent threads ranges from $1$ to $4$ for *Machine1*.

Similar experiments have been performed for *Machine2*. In this case the CPU frequency varied between $1600$ MHz (i.e., its minimum value) and $\{1600, 2500, 3401\}$ MHz. The number of parallel threads ranged from $1$ to $8$.

For the purpose of these experiments, the number of threads generated by the benchmark has been set equal to the maximum number of threads available for each configuration considered. Moreover, for each different configuration, $20$ benchmark executions have been performed.

Figure 3.1a depicts the average power consumed by *Machine1*, with $95\%$ confidence intervals, to process the workload generated by *Sunflow* benchmark under different parameters configurations (i.e., different values of CPU frequency and number of concurrent threads). As expected, the power linearly increases with respect to the maximum CPU clock frequency. Such increment grows according to the number of parallel threads used; for instance, using only one thread, the increment from $1600$ to $3300$ MHz is about $35\%$ (from 35 to 47 watt), while using four threads the increment is almost $60\%$ (40 to 63 watt).

Instead, when varying the number of parallel threads, the power does not increases linearly; indeed, it follows a step behavior with a gap of nearly $10$ watt from the second to the third configuration. In fact, the gap analysis is particularly interesting between configuration $2-1$ and configuration $2-2$, when the number of logical threads is the same, but in the first case both the threads are on the same physical core, whereas in the second one each thread is served by a different core.

An analogous system behavior has been observed studying *Machine2*; the results are depicted in Figure 3.1b, where average power consumption values with $95\%$ confidence intervals are represented. In this case, a larger

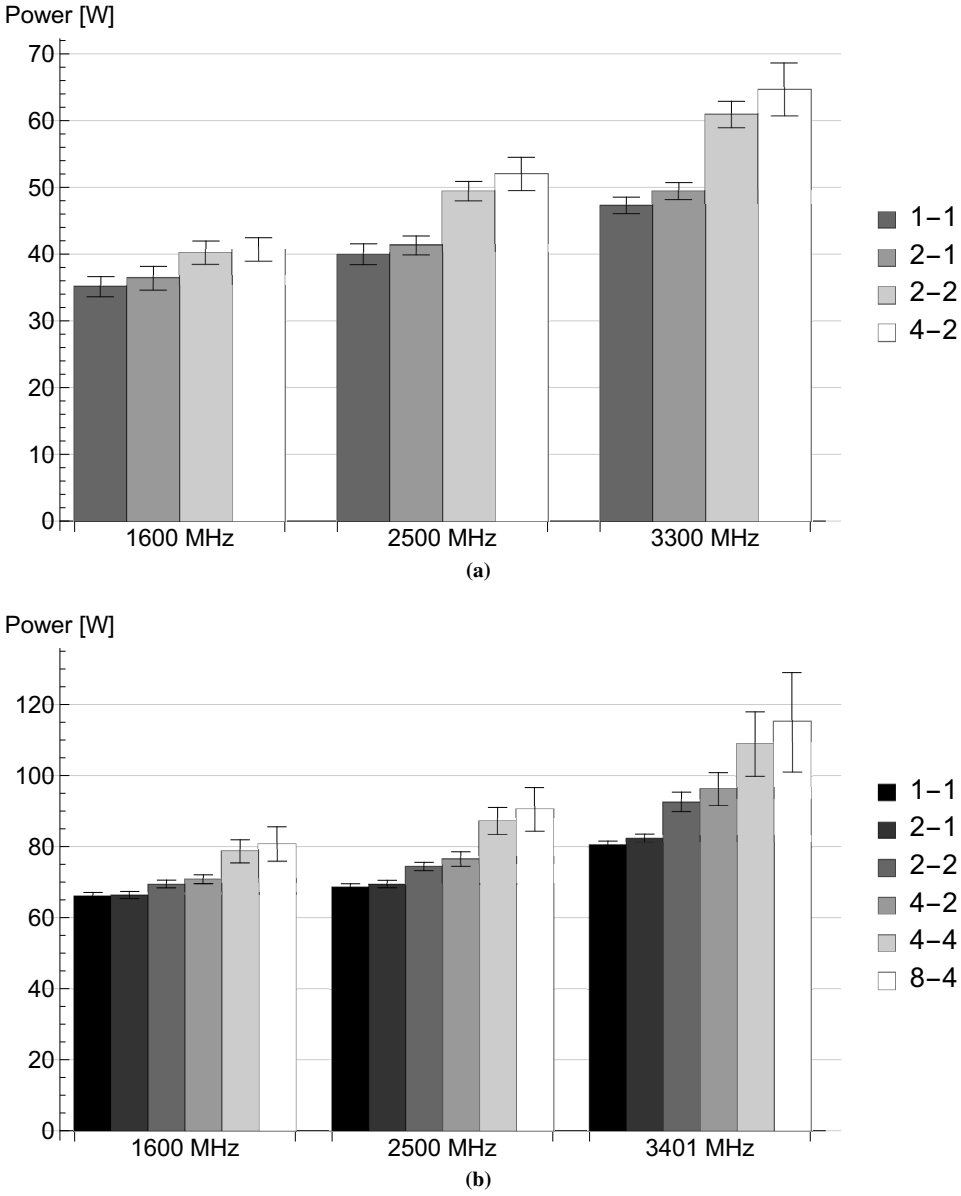



**Figure 3.1:** *Power required by (a) Machine1 and (b) Machine2 to execute a CPU-bound application, varying CPU speed and the number of available logical threads. In the legend, $x - y$ denotes a configuration with $x$ logical threads running on $y$ physical cores. Note that, $x$ may be twice larger than $y$ due to SMT.*

gap is registered while varying the number of active cores from two to four.

## 3.4 Power modeling

The linear and non-linear power models proposed in [49] and already described in Eqs. (2.7) and (2.8), respectively, have been adopted in this chapter as the starting points to provide a new power consumption model that accounts also for DVFS and SMT.

In order to investigate how to accurately model the system's power consumption, $P_{idle}$ and $P_{busy}$ have been estimated for both *Machine1* and *Machine2* observing the collected samples. Instead, the values of the calibration parameter $r$ has been derived from the collected observations by a fitting procedure.

The fitting procedure has been performed by the DEPSO evolutionary algorithm proposed by Zhang and Xie in [150]. As said by the authors, the DEPSO algorithm integrates the Differential evolution (DE) and Particle swarm optimization (PSO) techniques; while the PSO tries to iteratively improve a solution based on a given measure of quality, the DE algorithm introduces some mutations on the results computed by the PSO to maintain population diversity.

To prevent over-fitting, half of the collected measures are used to test the model and compute the value of the parameters, whereas the other half is used for computing the current value of the mean absolute percentage error (MAPE). MAPE is defined as:

$$MAPE = \frac{1}{N} \sum \left| \frac{A_t - F_t}{A_t} \right| \tag{3.2}$$

where $A_t$ is the actual value and $F_t$ is the estimated one.

Application of the standard power models given in Eqs. (2.7) and (2.8) to estimate the power consumption of the experiments described in Section 3.3 provides results with a large MAPE (i.e., $\text{MAPE}_{\text{Eq(2.7)}} \approx 42\%$ and $\text{MAPE}_{\text{Eq(2.8)}} \approx 41\%$ for the quad-core CPU). Indeed, for all the experiments the CPU utilization observed with `iostat` command is closed to $100\%$ and it is not scaled based on the real resources utilization (i.e., the number of active threads an the real CPU speed). In this case, according to Eqs. (2.7) and (2.8), the server's power consumption is always maximum.

In order to alleviate that problem, the CPU utilization is weighted first with the fraction of active cores, then with the portion of active threads.

Thus, the new equations adopted to compute CPU utilization are:

$$U_{cpu}(C_{cr}) = \frac{C_{cr}}{T_{cr}} \cdot U_{cpu} \tag{3.3}$$

and

$$U_{cpu}(C_{th}) = \frac{C_{th}}{T_{th}} \cdot U_{cpu} \tag{3.4}$$

where $C_{cr}$ ($C_{th}$) is the number of active cores (threads) and $T_{cr}$ ($T_{th}$) is the total number of cores (threads) that the CPU is capable to concurrently run.

Substituting the utilization as computed in Eqs. (3.3) and (3.4) into the power models defined in Eqs. (2.7) and (2.8), it is possible to obtain more accurate estimates of the system power consumption. For instance, applying the non-linear power model to the quad-core case, $\text{MAPE}_{\text{Eq(3.3)}} \approx 14\%$ and $\text{MAPE}_{\text{Eq(3.4)}} \approx 12\%$.

The MAPE may be reduced also taking into account the maximum frequency at which the CPU is allowed to work while estimating the server's power consumption. For this purpose, the CPU utilization is weighted also by the ratio of the frequency currently allowed to its maximum value (i.e., 3300 MHz and 3401 MHz for dual-core and quad-core CPU, respectively). Thus, the CPU utilization is derived either as:

$$U(\mathit{fr}, C_{cr}) = \frac{C_{cr}}{T_{cr}} \frac{\mathit{fr}}{\mathit{fr}_{max}} \cdot U_{cpu} \tag{3.5}$$

or as:

$$U(\mathit{fr}, C_{th}) = \frac{C_{th}}{T_{th}} \frac{\mathit{fr}}{\mathit{fr}_{max}} \cdot U_{cpu} \tag{3.6}$$

where, *fr* is the CPU speed allowed, and $\mathit{fr}_{max}$ is its absolute maximum value.

Substituting Eqs. (3.5) or (3.6) into the linear and non-linear power models proposed by Fan et. al in [49] allows to further decrease the MAPE for the quad-core case between $5\%$ and $9\%$

Finally, a scaling factor that accounts for available cores and threads, and the maximum CPU frequency allowed, may be adopted in order to further decrease the MAPE, thus obtaining more accurate power consumption estimates.

The scaling factor is defined as a function of *operating parameters* $O = \{C_{th},\ C_{cr},\ \mathit{fr}\}$ which may be set by the applications, and *machine parameters* $M = \{T_{th},\ T_{cr},\ \mathit{fr}_{max}\}$ that are fixed and depend on the hardware characteristics of the server considered. The equation:

$$\Delta(O, M) = \left(\frac{C_{th}}{T_{th}}\alpha_{th} + \frac{C_{cr}}{T_{cr}}\alpha_{cr}\right)\left(\frac{\mathit{fr}}{\mathit{fr}_{max}}\right)^{\eta} \tag{3.7}$$

defines the scaling factor, where $\alpha_{th}$, $\alpha_{cr}$ and $\eta$ are calibration parameters evaluated during the validation phase through the DEPSO algorithm.

Thus, CPU utilization is scaled, taking into consideration the scaling factor derived from Eq. (3.7), as:

$$U_{cpu}(O, M) = \Delta(O, M) \cdot U_{cpu} \tag{3.8}$$

and the server power consumption depending on the current operating conditions and the system's feature, $P(O, M)$, is derived substituting $U_{cpu}$ in Eqs. (2.7) and (2.8) with the scaled utilization $U_{cpu}(O, M)$ computed in Eq. (3.8) as:

$$P(O, M) = P_{idle} + (P_{busy} - P_{idle}) \cdot U_{cpu}(O, M) \tag{3.9}$$

and

$$P(O, M) = P_{idle} + (P_{busy} - P_{idle}) \cdot (2U_{cpu}(O, M) - U_{cpu}(O, M)^r) \tag{3.10}$$

for the liner and non-linear cases, respectively.

Usage of scaling factor $\Delta(O, M)$ to derive the real CPU utilization reduces the MAPE between $3\%$ and $4\%$ in the quad-core case.

The results obtained through the normalized utilization for both *Machine1* and *Machine2* are depicted in the Q-Q plots shown in Figure 3.2. For the sake of clarity, only the non-linear power models obtained using CPU utilization derived as in Eqs. (3.5) and (3.8) are represented. The Q-Q plots depict the power estimates derived through the considered power model against the power measures collected by the *Yokogawa WT210 digital power meter*. If the two values are the same, the point lies on the bisection line. As expected, analyzing the results obtained for the same machine with different CPU utilization models (i.e., $P(U_{cpu}(\boldsymbol{fr}, C_{cr}))$ and $P(U_{cpu}(O, M))$, it may be seen that the power estimates obtained using the scaling factor $\Delta(O, M)$ are more accurate than those obtained considering only the CPU speed and the number of active cores.

For further details the reader may refer to Tables 3.1 and 3.2, where the values of the calibration parameters used for the power model considered and its MAPE are reported.

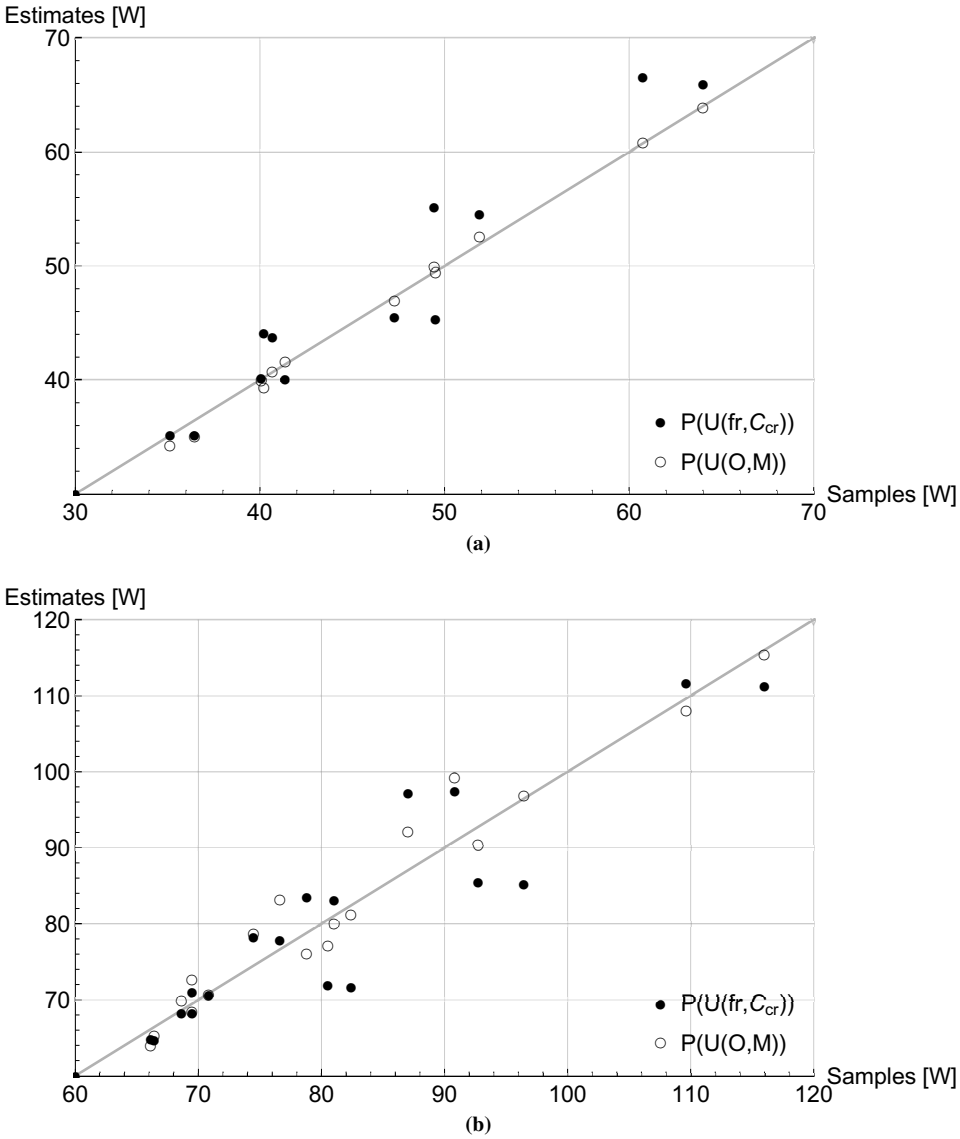



**Figure 3.2:** *Comparison between the power samples and the power estimates for (a) Machine1 and (b) Machine2. The estimates have been obtained using the CPU utilization as defined in Eqs. (3.5) and (3.8).*

**Table 3.1:** *The calibration parameters and MAPE for each power model are provided in this table. The machine considered is Machine1. Columns from* 1 *to* 3 *provide information about the power model, the CPU utilization definition and the equation where it is defined; the value of all the calibration parameters are given in the columns from* 4 *to* 7*; finally, last column states the MAPE.*

| Power Model | $U_{cpu}$ def. | $U_{cpu}$ Eq. | $r$ | $\alpha_{cr}$ | $\alpha_{th}$ | $\eta$ | MAPE |
|---|---|---|---|---|---|---|---|
| Linear | $U_{cpu}$ | — | — | — | — | — | 46.181% |
| Non-Linear | $U_{cpu}$ | — | 0 | — | — | — | 45.318% |
| Linear | $U_{cpu}(C_{cr})$ | (3.3) | — | — | — | — | 24.432% |
| Non-Linear | $U_{cpu}(C_{cr})$ | (3.3) | 0.509 | — | — | — | 20.633% |
| Linear | $U_{cpu}(C_{th})$ | (3.4) | — | — | — | — | 17.657% |
| Non-Linear | $U_{cpu}(C_{th})$ | (3.4) | 0.793 | — | — | — | 17.533% |
| Linear | $U_{cpu}(fr, C_{cr})$ | (3.5) | — | — | — | — | 7.869% |
| Non-Linear | $U_{cpu}(fr, C_{cr})$ | (3.5) | 0.809 | — | — | — | 5.517% |
| Linear | $U_{cpu}(fr, C_{th})$ | (3.6) | — | — | — | — | 9.180% |
| Non-Linear | $U_{cpu}(fr, C_{th})$ | (3.6) | 1.225 | — | — | — | 8.502% |
| Linear | $U_{cpu}(O, M)$ | (3.8) | — | 0.699 | 0.280 | 1.379 | 1.845% |
| Non-Linear | $U_{cpu}(O, M)$ | (3.8) | 2.918 | 0.400 | 0.157 | 1.670 | 1.072% |

**Table 3.2:** *The calibration parameters and MAPE for each power model are provided in this table. The machine considered is Machine2. Columns from* 1 *to* 3 *provide information about the power model, the CPU utilization definition and the equation where it is defined; the value of all the calibration parameters are given in the columns from* 4 *to* 7*; finally, last column states the MAPE.*

| Power Model | $U_{cpu}$ def. | $U_{cpu}$ Eq. | $r$ | $\alpha_{cr}$ | $\alpha_{th}$ | $\eta$ | MAPE |
|---|---|---|---|---|---|---|---|
| Linear | $U_{cpu}$ | — | — | — | — | — | 42.747% |
| Non-Linear | $U_{cpu}$ | — | 0 | — | — | — | 41.632% |
| Linear | $U_{cpu}(C_{cr})$ | (3.3) | — | — | — | — | 15.920% |
| Non-Linear | $U_{cpu}(C_{cr})$ | (3.3) | 0.724 | — | — | — | 14.211% |
| Linear | $U_{cpu}(C_{th})$ | (3.4) | — | — | — | — | 12.306% |
| Non-Linear | $U_{cpu}(C_{th})$ | (3.4) | 1.009 | — | — | — | 12.298% |
| Linear | $U_{cpu}(fr, C_{cr})$ | (3.5) | — | — | — | — | 5.239% |
| Non-Linear | $U_{cpu}(fr, C_{cr})$ | (3.5) | 0.945 | — | — | — | 5.132% |
| Linear | $U_{cpu}(fr, C_{th})$ | (3.6) | — | — | — | — | 8.303% |
| Non-Linear | $U_{cpu}(fr, C_{th})$ | (3.6) | 1.437 | — | — | — | 6.727% |
| Linear | $U_{cpu}(O, M)$ | (3.8) | — | 0.753 | 0.348 | 1.244 | 3.862% |
| Non-Linear | $U_{cpu}(O, M)$ | (3.8) | 1.510 | 0.581 | 0.505 | 1.888 | 3.177% |

CHAPTER 4

# Energy per time-unit of execution: a multi-class energy metric

## 4.1 Motivation

Besides proposing new techniques to decrease the data-centers' energy consumption, scientists also need to determine the infrastructure greenness level. Thus, green performance metrics have been proposed in literature [140] in order to qualitatively and quantitatively define that data-centers' parameter. In fact, the energy metrics are crucial in developing green data-centers for many reasons: first of all, they let the designers clearly determine the energy efficiency of the infrastructure; second, they enable the comparison of the data-centers from the energy consumption perspective; finally, they provide guidelines to designers and service providers to improve the performance of the data-centers.

Nevertheless, the energy metrics have been poorly studied and some data-centers' features have never been taken into account while developing the currently available ones. It is the case of multi-class workload, although differences and interactions in power consumption have been experimentally observed in systems that concurrently run multiple types of application [27].

For instance, a data-center may concurrently execute applications with different bottlenecks, such as scientific computing applications (CPU-bound), in-memory databases (memory-bound), MapReduce processes (I/O intensive) and video transcoding requests (GPU-bound).

The system's component to which the application is bound depends on the class of the application. That may generate bottlenecks which prevent the fully utilization of each resource, thus producing a waste of power due to the underutilization (i.e., idle period) that all the no-bottleneck resources are subject to.

For these reasons, a metric to accurately study the power and energy consumptions of a data-center must take into account both the different kind of applications that is processed by the system and the components that are exploited to serve each request.

Following this idea, the *energy per time-unit of execution* metric (abbreviated as EX) accounts for the class of each job and may take into consideration all the components of a server and their energetic relationship with the other system's resources.

## 4.2 A review of the available energy metrics

Wang and Khan [140] analyzed and classified the available energy metrics and provided general guidelines for data-center green metrics development. In fact, they proposed the five following criteria to make a data-centers energy metric efficient: *i)* it must adapt to each type of data-center (e.g., multi-tier data-centers, data-centers with redundancy, internal data-centers, etc.); *ii)* it should takes into account the whole system and not amplify the effects of few components; *iii)* it must consider the data-center's typical working conditions; *iv)* its implementation in current system must be as easy as possible in order to facilitate its dissemination and acceptance; *v)* also its utilization should be as simple as possible to spread the new energy metric adoption.

Moreover, they proposed to organize the green metrics based on the following characteristics:

- **point of view** of either the user or the service provider. Indeed, while the users are usually interested in the environmental impact of their applications, the system administrator cares for the greenness level of the data-center and its components;

- **measure grain**, since the energy metric may measure the energy consumption to different levels, from the finest (e.g., per bit, per instruc-

tion, etc.) to the coarsest grains (e.g., per racks, per data-center);

- **number of dimensions** analyzed, thus being single-dimension or *basic* when only a parameter of the system is taken into consideration (e.g., dioxide carbon emissions, system temperature, etc.), and multi-dimension or *extended* when the measure depends on different variables, such as the *Total cost of ownership* (TCO) that accounts for purchasing, operating and maintenance costs.

Greenhouse gases emission, humidity, thermal and power/energy metrics are classified as *basic metrics*. Indeed, they take into account only specific parameters in order to evaluate the data-centers energy efficiency. For the purpose of this thesis, only the power/energy metric are deeply described in the following, whereas emissions, humidity and thermal metrics are not further investigated. In fact, due to the large amount of power and energy consumed by the data-centers, several power metrics have been proposed in literature to monitor and study this problem.

*Data-center infrastructure efficiency* (DCiE) and *Power usage effectiveness* (PUE) have been proposed in [8] and are widely accepted by the industries. PUE is computed as:

$$PUE = \frac{\textit{Total Facility Power}}{\textit{IT Equipment Power}} \tag{4.1}$$

where the *Total Facility Power* is the power required to make the whole data-center work (e.g., IT equipment, cooling, light, etc.) and the *IT Equipment Power* is the power needed for only the IT equipment (e.g., compute, storage, network, etc.). It can only be greater than or equal to one and represents the relationship between the power consumption of IT infrastructure and all the other facilities. Thus, if $PUE = 2.0$, it means that the data-center power consumption is two times greater than the power consumption of its IT components.

Instead, DCiE is defined as the reciprocal of PUE, thus:

$$DCiE = \frac{1}{PUE} = \frac{\textit{IT Equipment Power}}{\textit{Total Facility Power}} \tag{4.2}$$

When $DCiE = 0.5$ (i.e., $PUE = 2.0$), it means the IT equipment consumes half of the power required by the data-center.

Although PUE and DCiE are fundamentally the same, the authors claimed both of them are adopted to differently represent the energy distribution in data-centers. In Table 4.1 the typical values of PUE and DCiE [61], and the corresponding efficiency level[5], are shown.

[5] `http://www.42u.com/measurement/pue-dcie.htm`. Accessed: Jan. 15, 2018.

**Table 4.1:** *Some typical PUE and DCiE values for enterprise data-center. The last column reports the corresponding data-center efficiency.*

| PUE | DCiE | Efficiency |
|---|---|---|
| 1.2 | 83% | Very efficient |
| 1.5 | 67% | Efficient |
| 2.0 | 50% | Average |
| 2.5 | 40% | Inefficient |
| 3.0 | 33% | Very inefficient |

The evolution from PUE and DCiE is the *data-center energy productivity* (DCeP or DCP) [8]. Since it lets the user see the data-center as a black box (power and input data go into the box, while heat and outputs are returned), DCeP may help designer in investigating the data-center productivity that the authors claim to be difficult to determine. It is defined as:

$$DCeP = \frac{\text{Useful Work}}{\text{Total Facility Power}} \tag{4.3}$$

where Useful Work is the number of tasks that have been processed by the system during a given time window.

In order to investigate the energy-performance trade-off in data-centers, *Energy-Response time product* (ERP) and *Energy-response time weighted sum* (ERWS) have been proposed. For both these metrics, the performance parameter of the equation is the customer's response time. In particular, ERP (also know as *Energy-Delay product*) is defined as:

$$ERP = \mathbb{E}[E] \cdot \mathbb{E}[R] \tag{4.4}$$

where $\mathbb{E}[E]$ and $\mathbb{E}[R]$ are the expected values of energy consumption and response time, respectively. It has been used by Gonzalez and Horowitz [60] to study the energy dissipation in microprocessors and by Gandhi et al. [53] to analyze server farm management policies. Since ERP is defined as the product of two expectations, it may result in difficult analytical handling. For this reason, ERWS is another widely adopted metric to estimate the energy-performance trade-off of a system. Differently from *Energy-Response time product*, ERWS is defined as:

$$ERWS = w_1 \cdot \mathbb{E}[E] + w_2 \cdot \mathbb{E}[R], \qquad w_1, w_2 > 0 \tag{4.5}$$

where $w_1$ and $w_2$ are two constant weights used to change the impact of energy consumption and delay on the final output. This trade-off metric

was used by Albers and Fujiwara [2] to analyze scheduling problems in battery-operated systems to decrease the response time of all the jobs while keeping a low energy consumption. Instead, Wierman et al. [142] used ERWS to investigate how to scale the processing speed in order to improve the energy-performance trade-off of a computer communication system.

Although ERWS is better than ERP for analytical evaluation especially when used into non-linear optimization algorithms, it fails to effectively take into account the order of magnitude of the combined quantities. For instance, reducing $\mathbb{E}[R]$ from 1000 to 999 seconds produces on ERWS the same effect of decreasing the expected response time from 10 to 9 seconds, which in several cases may be unrealistic. Instead, ERP is able to distinguish between the two cases, recognizing the latter reduction is more advantageous than the former one.

Wang and Khan [140] also reviewed some *extended* metrics. As previously said, TCO is one of them since it accounts for capital and operational costs. In particular, the former includes the investment to purchase and build the infrastructure, the latter consists of all the monthly costs required to run the data-center. Other *extended* metrics combine some key data-center's indicators: for instance, it may consists of DCiE and the utilizations of server, storage, network and data-center.

Hereinafter ERP and ERWS are the main comparison measures that will be adopted while proposing new energy metrics and investigating available systems and applications. Indeed, they are easier to derive and to adapt to different levels of detail (e.g., single components, a server, the whole data-center) than the other power/energy metrics analyzed in this section.

## 4.3 Energy per time-unit of execution

Although the definition of a metric to estimate energy consumed per unit of work is relatively straightforward for systems with one type of workload, this is no longer the case when considering multi-class systems. In this section, an energy metric to study the energy-performance trade-off in multi-class systems is presented and it is derived starting from the single-class case. Moreover, while considering multi-class workloads, the metric also accounts for the relationship between each components in the case of models with more than one resource.

As a general remark, in the considered cases the proposed definitions do not depend on a specific power consumption model. Indeed, energy and performance metrics are first derived and then used as input parameters to study the energy-performance trade-off. Thus, according to the complexity

of the analyzed scenario, it is possible to use either a simple linear power model that evaluates the server's power consumption accounting only for CPU utilization, as the one described in Eq. (2.7), an advanced model that considers also contribution of other resources such as disks or memory, for example, those presented in Eqs. (2.10) and (2.11), or even a complex non-linear equation that accounts for DVFS and SMT such as the power model presented in Chapter 3.

### 4.3.1 Single-class workload

Let us start focusing on a single-class workload being executed by a single resource. If $C(T)$ is the number of jobs that the system has completed up to time $T$ and $E(T)$ is the total energy consumed by the system during the same time window. The *energy per job* at time $T$, $EJ(T)$, of the system considered is defined as follows:

$$EJ(T) = \frac{E(T)}{C(T)} \tag{4.6}$$

Referring to the average power consumption of the system as $P$ and to the system throughput as $X(T)$, where $X(T) = C(T)/T$, and recalling Eq. (2.6), the previous equation may be written as:

$$EJ(T) = \frac{P \cdot T}{X(T) \cdot T} = \frac{P}{X(T)} \tag{4.7}$$

Since the *utilization law* states $U = X(T) \cdot D$, where $D$ is the *service demand* (i.e. the average time a job uses the resource during its execution), the dependency on the time interval $T$ may be dropped and the average energy consumption per job is derived as:

$$EJ = D \cdot \frac{P}{U} \tag{4.8}$$

The previous expression is interesting especially when using power models that depends entirely on the utilization of a single resource (i.e., Eqs. (2.7) and (2.8), [49]), since it expresses the energy consumed by a job as a function of the utilization of the resource analyzed.

Eq. (4.8) may be extended and used to study the systems with multiple resources. Let us now call $P$ the power consumption of a system composed by $K$ resources, each one characterized by a power consumption $P_r$, and $X(T)$ the system throughput. Thus, $P$ is now computed as:

$$P = \sum_{r=1}^{K} P_r$$

and the *energy per job* of the system can be derived through the following equation:

$$EJ(T) = \frac{P}{X(T)} = \sum_{r=1}^{K} \frac{P_r}{X(T)} = \sum_{r=1}^{K} EJ_r(T) \tag{4.9}$$

where $EJ_r$ is the *energy per job* of resource $r$. Recurring again to the *utilization law* for multi-resource systems, $U_r = X(T) \cdot D_r$ (where $D_r$ is the service demand of resource $r$ and $U_r$ is its utilization), Eq. (4.9) is rewritten as:

$$EJ = \sum_{r=1}^{K} D_r \frac{P_r}{U_r} \tag{4.10}$$

### 4.3.2 Multi-class workload and single resource

When considering several classes of jobs, Eqs. (4.8) and (4.10) are not straightforward since different types of jobs may have different demand requirements. In order to cope with this issue, a single resource executing several classes of jobs is first considered, then the obtained results are extended in order to account also for multiple and possibly different resources.

It is worthwhile noticing that the problem of considering jobs with different execution times due to their different class is not specific of the *energy per job* metric, but it must be considered while evaluating energy-performance trade-off as a consequence of investigating systems with multi-class workload. Indeed, similar considerations would have been necessary also trying to extend classical metrics (i.e., ERP and ERWS) to a multi-class scenario.

Let us consider a system with $H$ classes, and refer to the number of completions of class $c$ in the time window $T$ as $C_c(T)$. The resource's power consumption depends on its global utilization that is computed as the sum of the utilizations of the resource while executing class $c$ tasks:

$$U = \sum_{c=1}^{H} U_c$$

When investigating systems with multi-class workload, it is not possible to compute the *energy per job* metric as in Eq. (4.6). Indeed, that measure is no longer fair since it does not take into account the different service demands of the jobs.

Thus, the *energy per time-unit of execution*, $EX$, is proposed to account for the different times required by each job to be processed and it is defined

as follows:

$$EX(T) = \frac{P \cdot T}{\sum_c D_c \cdot C_c(T)} = \frac{P}{\sum_c D_c \cdot X_c(T)} \tag{4.11}$$

where $D_c$ is the service demand of the class $c$ jobs and $X_c(T)$ is the class $c$ throughput of the resource considered. Using the *utilization law* for multi-class workloads, $U_c = X_c(T) \cdot D_c$, and dropping the dependency on $T$, Eq. (4.11) becomes:

$$EX = \frac{P}{\sum_c D_c \cdot X_c(T)} = \frac{P}{\sum_c U_c} = \frac{P}{U} \tag{4.12}$$

It is interesting to note that, when considering single-class systems, the following relationships between $EJ$ and $EX$ holds:

$$EJ = EX \cdot D \tag{4.13}$$

showing that the two metrics are related through a multiplicative constant that is the service demand of the resource observed.

### 4.3.3 Multi-class workload and multi resources

The extension of *energy per time-unit of execution* metric to multiple resources requires extra care. In particular, the system's components must be distinguished between *separable* and *inseparable* resources. *Separable* resources are energetically distinct and they can be enabled or disabled independently from the other resources. In this case, energy optimization can be performed by switching off unused resources. The replicas of tiers in a multi-tier application (e.g., database nodes or caching components) are an example of separable resources.

Instead, *inseparable* resources cannot be individually disabled. All the resources must be active for the system to work, even if they are not utilized in the considered scenario. For instance, these resources may represent the different always-on hardware components (e.g., disk, CPU, GPU, network), or services required to support the execution of the entire application (e.g., a database in a complex multi-tier web-service).

It is important to note that in some cases, depending on the specific configuration, the same resource (e.g., the disk) can be considered either *separable* if it is able to activate a low power consumption state, or *inseparable* if the OS has no way of turning the resource off. In other words, that is not an intrinsic characteristic of the resources, but depends on their capability to be separately disabled to preserve energy.

The *energy per time-unit of execution* in case of *separable* components can be computed as the sum of the same measure for each single resource, thus as:

$$EX_{Sep} = \sum_{r=1}^{K} EX_r = \sum_{r=1}^{K} \frac{P_r}{U_r} \tag{4.14}$$

However, Eq. (4.14) might not be fair when considering systems composed by *inseparable* resources since it tends to infinity if one of the component's utilization tends to zero.

Thus, a different definition of $EX$ is required for *inseparable* resources. In particular, it is defined as the ratio of system power consumption to the sum of all the components' utilization:

$$EX_{Insep} = \frac{\sum_{r=1}^{K} P_r}{\sum_{r=1}^{K} U_r} = \frac{P}{\sum_{r=1}^{K} U_r} \tag{4.15}$$

Although the denominator of Eq. (4.15) (i.e., the sum of the resources' utilization) might seem arbitrary, it may be related to specialized measures previously defined in the literature. Indeed, let us recall the definition of *normalized system throughput* for a system with multi resources and multi-class workload, given in [118]:

$$X' = \sum_{c} \frac{D_c}{D} X_c \tag{4.16}$$

where $D_c = \sum_{r=1}^{K} D_{rc}$ is the total demand of class $c$ jobs and the sum of all the service demands is $D = \sum_{c=1}^{H} \sum_{r=1}^{K} D_{rc}$. Through *utilization law*, the utilization of resource $r$ is defined as:

$$U_r = \sum_{c=1}^{H} U_{rc} = \sum_{c=1}^{H} X_c \cdot D_{rc} \tag{4.17}$$

Substituting Eq. (4.17) in Eq. (4.15), the following equation is obtained for $EX_{Insep}$:

$$EX_{Insep} = \frac{P}{\sum_{r=1}^{K} \sum_{c=1}^{H} X_c D_{rc}} = \frac{P}{\sum_{c=1}^{H} X_c \cdot D_c} \tag{4.18}$$

and multiplying both sides of Eq. (4.18) by $D$:

$$D \cdot EX_{Insep} = \frac{P}{X'} = EJ \tag{4.19}$$

since in the single-class case, the normalized system throughput as defined in Eq. (4.16) is equal to the system throughput. For the same reason, the relationships presented in Eqs. (4.13) and (4.19) are identical.

Furthermore, complex systems may be composed by several subsets of inseparable resources. In this case, the previous definitions given in Eqs. (4.14) and (4.15) may be combined to obtain a general expression to correctly evaluate the *energy per time-unit of execution* of jobs in multi-class systems.

For that purpose, let us partition the $K$ resources into $M$ groups. Each group $G_m = \{r_1, \ldots, r_{K_m}\}$ contains $K_m$ components. All the resources in a group $G_m$ are *inseparable*, while different groups $G_l$ and $G_m$ (with $l \neq m$) are *separable*. Thus, the *energy per time-unit of execution* for the entire system is defined as:

$$EX = \sum_{m=1}^{M} \frac{\sum_{r \in G_m} P_r}{\sum_{r \in G_m} U_r} \tag{4.20}$$

In this case, it is interesting to note that Eq. (4.20) may be reduced to Eq. (4.14) when $M = K$ and $G_m = \{m\}$. Instead, it is reduced to Eq. (4.15) when $M = 1$ and $G_1 = \{1, \ldots, K\}$.

Finally, it worths noting that, the three *energy per time-unit of execution* metrics, i.e., Eqs. (4.14), (4.15) and (4.20), can be used to study the energy consumption of a system. While using the same metric, it is possible to compare the energy efficiency of different systems or different system's configurations. Instead, the results obtained recurring to different metrics cannot be used for comparisons, since they do not account for all the system characteristics at the same way.

## 4.4 Numerical results

The *energy per time-unit of execution* metric introduced in the previous section is now applied to study a system with two resources and two classes and it is compared with ERP and ERWS metrics in order to compare their results. For this purpose, a system composed by two parallel workstations with the same characteristics of *Machine2* (described in Section 3.3.1) is considered. A queuing network is used to model the system analyzed and its representation is given in Figure 4.1.

The think time of the system in Figure 4.1 is zero (i.e., it is a batch system), thus the jobs do not spend time in the delay station. As said, a multi-class workload is considered and the users' requests are divided into class $A$ and class $B$ jobs.

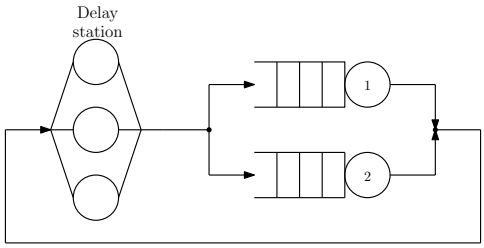



**Figure 4.1:** *The system considered to study the $EX$ metric.*

**Table 4.2:** *Average service demands of the web application adopted as case-study.*

| | Class $A$ | Class $B$ |
|---|---|---|
| Resource 1 | 3.84 s | 2.3808 s |
| Resource 2 | 0.96 s | 6 s |

Let us also assume the system can process a maximum of $N = 50$ requests simultaneously. Moreover, the administrator can tune the proportion of class $A$ and class $B$ requests into the system by appropriately varying the kind of jobs submitted to the system.

As done in Chapter 2, $\beta_A$ represents the percentage of class $A$ requests, and $\beta_B = 1 - \beta_A$ is the percentage of class $B$ jobs into the system. Thus, the population mix of the system is defined as $\vec{\beta} = (\beta_A, \beta_B)$.

The service demands of the to workstations for both class $A$ and class $B$ jobs are given in Table 4.2.

In order to study the population mix $\vec{\beta}$ that minimizes the energy consumption of the system while providing the best performance, the linear power consumption model described in Eq. (2.7) is adopted. As done for *Machine2* in Chapter 3, $P_{idle} = 59$ watt and $P_{busy} = 115$ watt for both the workstations considered.

The model here described has been analyzed with JMVA, the analytical solver of JMT [11], a comprehensive framework for performance evaluation, system modeling with analytical and simulation techniques, capacity planning and workload characterization studies, developed by Politecnico di Milano and Imperial College London.

The performance of the system shown in Figure 4.1 are depicted in Figure 4.2 against the percentage of class $A$ jobs into the system.

As visible, the best performance values are obtained when the population mix is $\vec{\beta} = (0.4, 0.6)$, that is the optimal population mix of the system

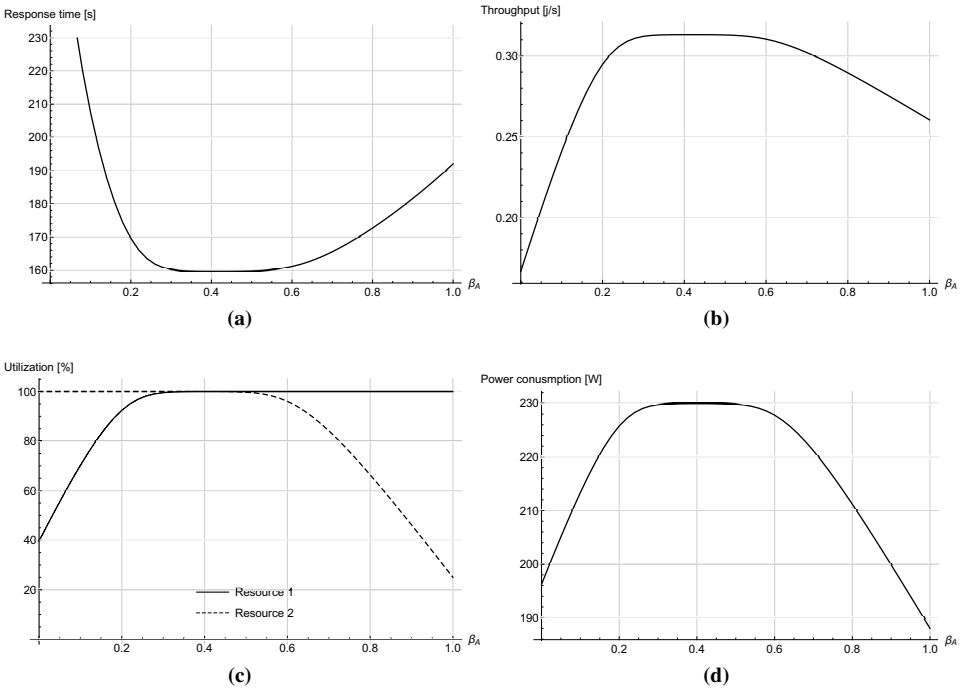



**Figure 4.2:** *The performance of the system considered in this section. Its response time is plotted in (a), the throughput in (b), the utilization of the two resources are depicted in (c) and the system power consumption in (d).*

considered, $\vec{\beta}^*$, and may be derived through Eq. (2.2). Thus, when the system works with 20 class $A$ and 30 class $B$ jobs, it provides the shortest system response time and the largest system throughput. Since the sum of the utilizations of each resource is maximum in $\vec{\beta}^*$, also the system power consumption (i.e., the sum of the power consumed by each resource) is maximum when the system works with the optimal population mix. Note that, as previously said in Chapter 2, all the $\vec{\beta}$ belonging to the common saturation sector – that is derived through Eq. (2.4) – make the system work with its best performance.

The power-performance trade-offs evaluated with ERP, ERWS and EX are depicted in Figure 4.3 as a function of $\beta_A$. In particular, since it was not possible to derive the system's energy consumption without knowing the time window for which the two workstations were turned on, the $\mathbb{E}[E]$ parameter in Eqs. (4.4) and (4.5) has been substituted by the expected power $\mathbb{E}[P]$ as in [53].

ERWS has been evaluated for different values of $w_1$ and $w_2$ weights; in fact, $w_1 = \{0.25, 0.50, 0.75\}$ and $w_2 = 1 - w_1$ in order to observe how ERWS changes when either the response time or the power consumption is the more important parameter for the service provider. Instead, Eq. (4.14) has been used to estimate EX, since the two parallel workstations may be independently shut down when they are not used.

The population mix for which the system has the best power-performance trade-off according to each metric is marked by a bullet. In order to compare all the different metrics, each value has been normalized between 0 and 1, dividing it by the maximum value estimated by the metric considered.

$EX_{Sep}$ is the only metric that identifies the optimal population mix as the one which provides the best power-performance trade-off. As said, in that point the system works with its minimum response time and maximum power consumption. However, if the system may complete the same amount and proportion of jobs in a shorter time, its energy consumption could be lower as said in Section 2.2.1.

Moreover, if the impact of power consumption is lower than the one of response time (i.e., ERWS with $w_1 = 0.25$ and $w_2 = 0.75$), the best operating point estimated by ERWS is very close to the one given by $EX_{Sep}$.

Finally, ERP and the other configurations of ERWS identify $\vec{\beta} = (1, 0)$ as the population mix with the best power-performance trade-off.

It is important to note that, in the case here analyzed, the computation of the total amount of time to serve a given number of jobs is not a trivial task. Indeed, recurring to the throughput definition (i.e., $X = C/T$, where $X$ is

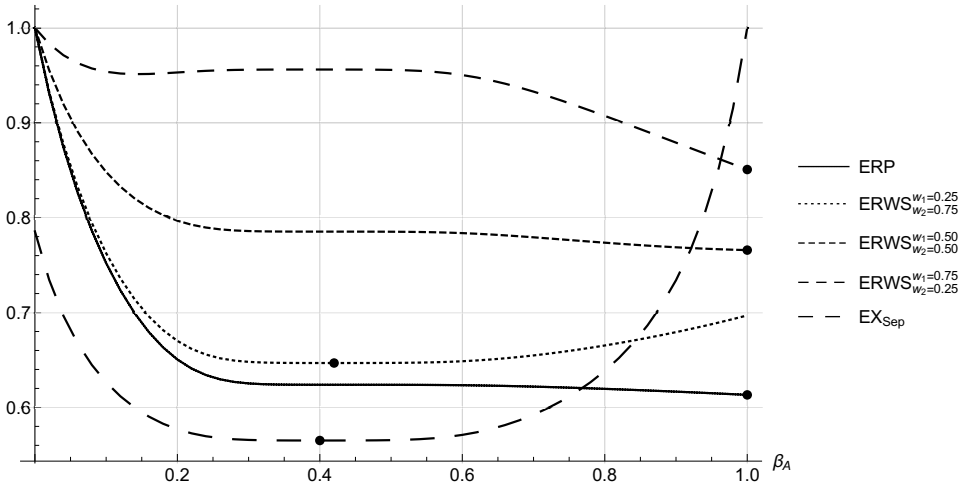



**Figure 4.3:** *The system's energy-performance trade-off evaluated with different metrics and normalized over the maximum value estimated by each metric. The minimum value of each curve, i.e., the best operating point proposed by the metric considered, is marked with* •.

the throughput and $C$ is the number of completed task in time $T$) to derive the time required to complete $N$ requests would be misleading, since the class of each processed job is not taken into account. That is exactly the problem which is tackled in this chapter, while introducing the *energy per time-unit of execution* metric.

# Part II

# Pool depletion systems and Optimal population mix strategy

CHAPTER 5

# Pool depletion systems

## 5.1 Motivation

The pervasiveness of Big Data applications in organizations is occurring at a surprising high speed. Successful companies must adopt these new technologies in order to keep their advantage over competitors.

One of the most important characteristics of this new paradigm is the large size of data that must be processed in a reasonable amount of time. To address the resulting performance problem, the Hadoop MapReduce technology has been proposed [38, 72] originally by Google. Its operational concept is based on distributed computing and parallelism. Initially, the input data is split into blocks that are processed in parallel by a large number of tasks generated by each job during the Map phase. In the following Reduce phase, newly created tasks process in parallel the intermediate results of the Map phase producing the final output of the job. The completion of a job may require one or more cycles of Map and Reduce phases. Typically, each phase can take hours, or even days, to complete due to the significantly large data sizes and the consequent high number of tasks to be executed in parallel.

Thus, the analysis of resource consumption behavior during the execu-

tion of a MapReduce job shows a pattern that is repeated in each phase: generation of a large number of tasks, followed by their parallel execution.

Also other current real life problems adopt the same execution pattern; for example, it is the case of video content providers, such as YouTube or Netflix, that often need to transcode a huge pool of videos [41] to multiple formats suitable to be sent and played by several different devices (e.g., smart-phones, smart-TVs, tablets, etc.).

The execution time of parallel tasks is deeply affected by two factors that are related to the characteristics of the tasks and the architecture of the computing infrastructure. While the first factor concerns the resource requirements of the tasks, since they may saturate different resources during their execution and the bottleneck of the system may migrate, the second one regards the characteristics of the system's components.

In fact, in cloud infrastructures, the resources that are dynamically allocated to the tasks may be heterogeneous and have different capacities (e.g., computational power, storage size and speed, etc.). Furthermore, these physical systems typically have a limitation on the number of tasks that may be concurrently processed. This constraint – continuously reached in this kind of applications due to the high number of tasks – is required for performance control on response time.

As a function of the mix of tasks in execution in a single server (referred to as *subsystem*), the time required by parallel execution of the tasks may be in some cases extremely inflated due to the bottlenecks migration.

The variability introduced in the execution time of the tasks may have a large impact on the completion time of the whole job. To cope with this problem, the tasks admission policy in each subsystem plays a fundamental role.

In order to study the performance and behavior of such applications – where a huge and fixed number of tasks, referred to as *pool*, waits to be admitted for execution in a set of service centers with limited capacity – *Pool depletion systems* are proposed and described in this chapter. The same framework is adopted to investigate the scheduling strategies able to decrease the time required for the completion of all the tasks initially allocated to the pool (referred to as *depletion time*).

In particular, the *Optimal population mix* policy is proposed and evaluated. It consists in scheduling the tasks execution in a way such that the optimal population mix (described in Section 2.1) of the considered system is exploited to minimize the depletion time.

Note that, several scheduling strategies have been described in literature to deal with systems that serve a large number of tasks. Each policy

may have different objectives: *i*) to minimize the execution time of each single task; *ii*) to optimize the system utilization exploiting the jobs allocation on each resource, *iii*) to minimize the execution time of a single job that is served by all the resources of the system, as may happen in scientific applications. *Case i* has been deeply analyzed in literature: FCFS, JSQ, MaxWeight and Completely Fair Scheduler strategies are examples of scheduling policies used in that case. Differently from *Case i*, in the problem considered in this chapter the minimization of task execution time does not necessarily minimize the time required for job completion. Also *Case ii* has been studied in depth; for example, Fair and Capacity schedulers [82] are adopted by Hadoop to allocate resources in order to improve system utilization when executing jobs possibly from multiple tenants. The objective of the problem here considered is described by *Case iii*.

## 5.2 System description

A Pool depletion system is a framework composed by a *pool* of independent tasks and one or more *subsystems* (i.e., servers that process the incoming requests). A *scheduler* placed between pool and subsystems lets each task in the pool be routed to one of the available servers; different scheduling strategies may be implemented to optimize the performance of the system. Let us assume all the tasks are created in the pool at the same time instant and then are sent to the available subsystems for their execution. In these systems, the most important parameter is the time required by the execution of all the tasks, i.e., the depletion time.

When considering MapReduce applications, the pool initially contains the Map tasks that, for simplicity, are assumed to be completed before starting the following Reduce phase (that is modeled with another Pool depletion system). Besides MapReduce applications, other examples of such type of workloads are video transcoding/analysis, applications of business analytics and NoSQL queries [26, 41].

In particular, we focus on jobs composed by two types of tasks, defined as class $A$ and class $B$. For instance, in multimedia stream applications each chunk is processed by a single task and the two classes may represent the computation of audio and video chunks, respectively. Instead, for what concern MapReduce-based applications, the two different classes may represent data retrieving from different tables and their joining to answer some users' queries.

Each subsystem is composed by two resources, denoted as *Res1* and *Res2* which, for example, may represent the CPU and the storage of a

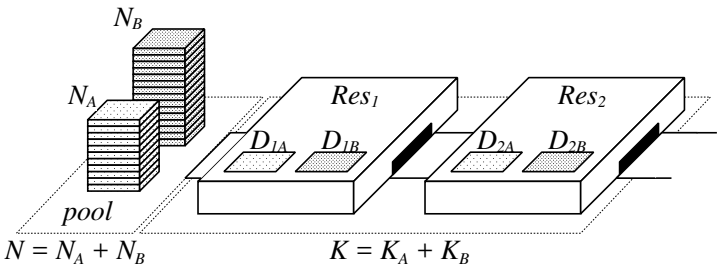



**Figure 5.1:** *Pool depletion system with only one subsystem.*

server. As said in previous chapters, the time required by a class $c$ task to be served by the resource $r$ is referred to as *service demand*, $D_{rc}$. The service demands characterize the workload in terms of total processing requirements to the resources and, in this chapter, they are assumed to be exponentially distributed. The two resources satisfy the assumptions of BCMP theorem: they can work either in *processor sharing*, or in *first come first served* with all the requests of identical service time, but possibly with different visit ratio. We assume that they execute the concurrent tasks according to a processor sharing queuing discipline: all the tasks are processed by the resources with a service rate proportional to the current number of tasks in service.

The number of tasks initially in the pool is denoted as $N = N_A + N_B$, where $N_A$ and $N_B$ are the number of class $A$ and class $B$ tasks, respectively. The system can execute no more than $K \leq N$ tasks concurrently. This limitation, that prevents all the $N$ tasks to be executed in parallel, may be used to model constraints on memory occupancy or restriction to comply with the SLAs. The system is allowed to execute $K_A$ tasks of class $A$ and $K_B$ of class $B$, such that $K = K_A + K_B$; note that, $K_A \leq N_A$ and $K_B \leq N_B$.

As soon as a task is completed, another task of the same class starts being processed by the subsystem. When all the tasks of a class are completed, the system allows the tasks of the other class to enter the subsystems until their capacity $K$ is reached. Figure 5.1 gives a visual representation of a Pool depletion system with only one subsystem.

Figure 5.2 shows the temporal evolution of a Pool depletion system with only one subsystem. In particular, the number of tasks concurrently processed into the subsystem and their proportions are depicted.

Initially, $K$ out of $N$ tasks immediately starts being processed by the first resource of the system. As long as there are tasks of both classes in

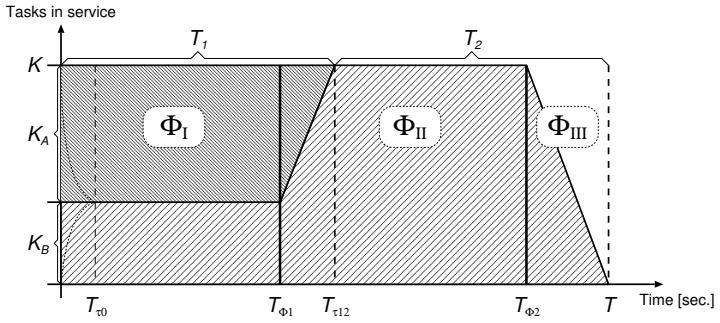



**Figure 5.2:** *Temporal evolution of the number of tasks processed by a Pool depletion system. The initial instant of each phase is shown on the x-axis.*

the pool waiting to be executed, the number of tasks into the subsystem is always $K = K_A + K_B$. This phase is addressed as $\Phi_{\mathrm{I}}$: after a short initial transient period, denoted with $T_{\tau 0}$, the system behaves as a closed queuing model with $K_A$ and $K_B$ customers, since whenever the task of a class leaves the subsystem, it is immediately replaced by another one of the same type.

At time $T_{\Phi 1}$ the tasks in the pool of one of the two classes are finished and there are not other tasks of that class which can enter the subsystem (in Figure 5.2, class $A$ tasks). At this time the system starts replacing the tasks of the exhausted class with the ones of the other class in order not to underutilize the subsystem; phase $\Phi_{\mathrm{II}}$ begins. Also in this case, after an initial transient period in which all the remaining tasks of the exhausted class are served, the system behaves as a closed queuing model with $K$ customers (in Figure 5.2 they are all class $B$ customers). The end of the transient period is denoted as $T_{\tau 12}$.

If the pool is empty, the subsystem begins to execute a decreasing number of tasks since they cannot be replaced by new ones when they are completed. The beginning of this last phase is denoted as $T_{\Phi 2}$, whereas the whole period of time in which the server is working with less than $K$ tasks is referred to as phase $\Phi_{\mathrm{III}}$. The job has been completely executed at time $T$, i.e., the depletion time, when all its tasks have been processed. Indeed, the depletion time is the total time required to complete the execution of all the $N$ tasks initially in the pool.

## 5.3 Modeling a Pool depletion system

Two different models have been adopted to investigate the Pool depletion systems' behavior, and they are presented in this section. In particular,

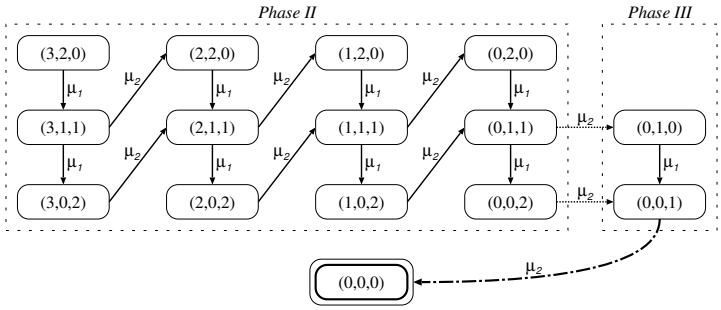



**Figure 5.3:** *CTMC describing a Pool depletion system with single-class workload and one subsystem, when $N_A = 5$ and $K_A = 2$.*

continuous time Markov chain (CTMC) is used to analytically study the new framework, but only very simple systems can be analyzed due to the state space explosion problem. Thus, a multi-formalism model has been proposed in order to investigate more complex systems.

### 5.3.1 Continuous time Markov chain

Pool depletion systems with only one subsystem and single-class or multi-class workload have been modeled with CTMC. Although the system description provided in Section 5.2 for the framework considered seems to be very simple, the underlying Markov process is characterized by many asymmetries that makes its analysis a bit involved. To simplify the presentation, a simple single-class example with fixed parameters is initially presented, then it is extended to the two-class case.

#### Single-class model

Let us consider a single-class Pool depletion system with $N = N_A = 5$ tasks to be completed, and only one subsystem with capacity $K = K_A = 2$, i.e., the number of tasks that are allowed to be executed at the same time. The corresponding CTMC is shown in Figure 5.3, and its state is identified by the tuple: $(n_{OA}, n_{1A}, n_{2A})$, where $n_{OA}$ is the number of tasks that are waiting in the pool to enter the subsystem, $n_{1A}$ is the number of tasks in resource *Res1* and $n_{2A}$ is the number of tasks in resource *Res2*. Note that $n_{1A} + n_{2A} \leq K$.

The loading phase is ignored since it is negligible, thus $K$ tasks are immediately admitted in resource *Res1*. For this reason, the initial state of the CTMC is $(n_{OA} - n_{1A}, n_{1A}, 0) = (3, 2, 0)$.

Let us call $\mu_1 = 1/D_{1A}$ the rate at which tasks leaves *Res1*, and $\mu_2 =$

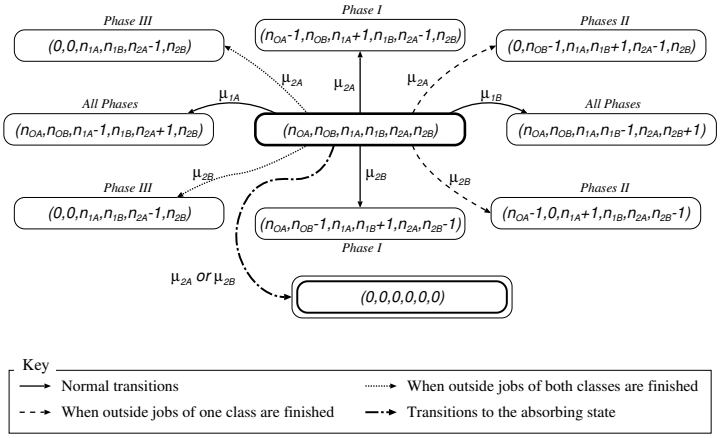



**Figure 5.4:** *Portion of the CTMC that describes the Pool depletion system with single subsystem and two-class workload.*

$1/D_{2A}$ the rate at which tasks are served by *Res2*. The tasks always leave from *Res1* to *Res2* at rate $\mu_1$, producing a transition from $(n_{OA}, n_{1A}, n_{2A})$ to $(n_{OA}, n_{1A} - 1, n_{2A} + 1)$.

The effect of tasks completion at resource *Res2* is different depending on whether there are other tasks waiting to be processed ($n_{OA} > 0$, phase $\Phi_{\text{II}}$ in Figure 5.2) or not. In the former case, the system performs a transition from state $(n_{OA}, n_{1A}, n_{2A})$ to state $(n_{OA} - 1, n_{1A} + 1, n_{2A} - 1)$ at rate $\mu_2$, since whenever a task exits the system from resource *Res2*, one of the waiting tasks immediately enters the subsystem and is served by *Res1*. Instead, if the tasks waiting in the pool are finished ($n_{OA} = 0$, phase $\Phi_{\text{III}}$ in Figure 5.2), the subsystem starts working with one less task, transitioning from state $(0, n_{1A}, n_{2A})$ to state $(0, n_{1A}, n_{2A} - 1)$ at rate $\mu_2$. When the last task has been executed, the system is in the absorbing state $(0, 0, 0)$.

#### Multi-class model

Figure 5.4 shows the basic transition structure of the CTMC underlying a two-class Pool depletion system with a single subsystem. For the sake of clarity, only outgoing arcs are shown.

When considering a Pool depletion system processing a two-class workload, each state is characterized by a six components tuple, $(n_{OA}, n_{OB}, n_{1A}, n_{1B}, n_{2A}, n_{2B})$, that represents the number of tasks waiting in the pool and being executed at *Res1* and *Res2* for both the classes.

If $n_{1A} + n_{1B} > 0$, then the tasks are processed by *Res1*; thus, the transition may be either to state $(n_{OA}, n_{OB}, n_{1A} - 1, n_{1B}, n_{2A} + 1, n_{2B})$ or to

state $(n_{OA}, n_{OB}, n_{1A}, n_{1B}-1, n_{2A}, n_{2B}+1)$ at rate:

$$\mu_{1c} = \frac{n_{1c}}{n_{1A}+n_{1B}} \cdot \frac{1}{D_{1c}}, \quad c \in \{A, B\} \tag{5.1}$$

where the ratio $n_{1c}/(n_{1A}+n_{1B})$ implements the processor sharing policy used by the resource.

Instead, the completion of a task at resource *Res2* can trigger four different behaviors, each one leading to a different pattern for the next state. Let us focus on class $A$ tasks; the case for class $B$ is symmetrical.

If class $A$ tasks are still waiting in the pool to be served (i.e., $n_{OA} > 0$, phase $\Phi_{\mathrm{I}}$ in Figure 5.2), the system will allow a new class $A$ task to start its execution. This leads the system to state $(n_{OA}-1, n_{OB}, n_{1A}+1, n_{1B}, n_{2A}-1, n_{2B})$ and it is represented in Figure 5.4 by the continuous arcs.

If there are no more class $A$ tasks in the pool (i.e., $n_{OA} = 0$) but at least one class $B$ task is waiting to be processed (i.e., $n_{OB} > 0$, phase $\Phi_{\mathrm{II}}$ in Figure 5.2), thus the completion of a class $A$ task makes a class $B$ task enter the subsystem in order to exploit its capacity $K$. That leads the system to state $(0, n_{OB}-1, n_{1A}, n_{1B}+1, n_{2A}-1, n_{2B})$ and is represented by a dashed arc in Figure 5.4. Note that, during phase $\Phi_{\mathrm{II}}$, the components $n_{1A}$ and $n_{2A}$ become 0 after the transient period $T_{\tau 12}$, as shown in Figure 5.2.

If no more tasks are into the pool (i.e., $n_{OA} = 0$ and $n_{OB} = 0$, phase $\Phi_{\mathrm{III}}$ in Figure 5.2), then the subsystem starts depleting and serving less than $K$ tasks in parallel; this is represented by state $(0, 0, n_{1A}, n_{1B}, n_{2A}-1, n_{2B})$. The depletion phase, the one considered when the pool is empty, is represented in Figure 5.4 by a dotted arc.

Finally, when the last task is completed, the system reach the absorbing state $(0, 0, 0, 0, 0, 0)$. This is represented with a dash-dotted arc in Figure 5.4. As for *Res1*, due to the processor sharing scheduling algorithm, the service rate of *Res2* is:

$$\mu_{2c} = \frac{n_{2c}}{n_{2A}+n_{2B}} \cdot \frac{1}{D_{2c}}, \quad c \in \{A, B\} \tag{5.2}$$

#### Models analysis

In order to compute the depletion time, the well-known technique for evaluating the up to-absorption time [103] is applied. Let us consider the CTMC of the general model with absorbing state $(0, 0, 0, 0, 0, 0)$ and infinitesimal generator matrix $\mathbf{Q} = [q_{ij}]$, and call $W$ the set of non-absorbing states.

The mean time spent by the CTMC in state $i$ until absorption is defined

as

$$z_i = \int_0^\infty \pi_i(\tau) d\tau$$

where $\pi_i(\tau)$ is the unconditional probability of the CTMC being in state $i$ at time $\tau$. The row vector $\mathbf{z} = [z_i]$ satisfies the following equation:

$$\mathbf{z}\, \mathbf{Q}_W = -\pi_W(0) \tag{5.3}$$

where $\pi_W$ and $\mathbf{Q}_W$ are the transient probability vector and the infinitesimal generator matrix restricted to the non-absorbing states only, respectively. Following [103], the mean time to absorption of the CTMC, $T$, can be computed from the solution of Eq. (5.3):

$$T = \sum_{i \in W} z_i$$

If we call $P_i$ the average power consumed in state $i$, then the average total energy consumed by the system is:

$$E = \sum_{i \in W} z_i \cdot P_i$$

In a similar way, if we call $u_{ri}$ an indicator function that is true if a resource $r$ is used in state $i$, $\phi_i(X)$ the indicator function that is true when state $i$ belongs to phase $X \in \{\mathrm{I}, \mathrm{II}, \mathrm{III}\}$, and $m_i$ the number of tasks admitted in the subsystem in state $i$, then the average utilization of resource $r$, $U_r$, the average time $\Phi(X)$ spent in phase $X$, and the average number of tasks in the subsystem can be derived as:

$$U_r = \frac{1}{T} \sum_{i \in W} z_i \cdot u_{ri}\ , \quad \Phi(X) = \sum_{i \in W} z_i \cdot \phi_i(X)\ , \quad M = \frac{1}{T} \sum_{i \in W} z_i \cdot m_i$$

### 5.3.2 Multi-formalism model

A multi-formalism model is used to evaluate more complex configurations of the Pool depletion systems. Indeed, when values of $N$ and $K$ increase or multiple subsystems are considered, the CTMC model cannot provide any results in a reasonable time due to state space explosion.

Thus, a Pool depletion system may be described as shown in Figure 5.5, using a multi-formalism model consisting of Colored Petri Net (CPN) and multi-class fork-and-join queuing network. The workload of the model consists of two type of customers: the tokens, representing the colors of the Petri net, and the jobs, representing the requests to be executed by the

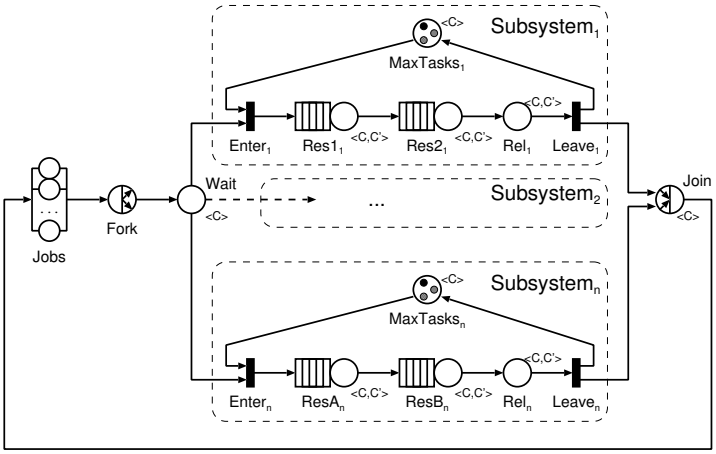



**Figure 5.5:** *The multi-formalism model used to study Pool depletion systems.*

**Table 5.1:** *Color-sets of the Petri net in the multi-formalism model.*

| Colour-set | Description |
|---|---|
| $< C >$ | Task class $C = \{A, B\}$ |
| $< C, C' >$ | Task class $C$, original class $C'$ |

queuing network. Each type of workload comprises two different classes of customers. CPN tokens are used to model the admission policy of the new tasks into each subsystem, while queuing network primitives are used to describe the service demands of the tasks. According to [62], the use of several formalisms allows exploiting the most appropriate modeling primitives to express the corresponding concepts in the most efficient and natural way.

The jobs are created in the delay station `Jobs`; note that, it is possible to focus on a single job that is continuously executed, without loss of generality, and the *think time* (i.e., the service time of the `Jobs` station) is set to $Z = 0$. When the job enters the `Fork` node, it is split into $N_A$ tasks of class $A$ and $N_B$ tasks of class $B$, with $N = N_A + N_B$. The tasks waiting to be executed are represented as tokens into place `Wait`: in this case, color class $< C >$ is used to remember the task class as an attribute of the token. Table 5.1 summarizes the color classes used in the model; in Figure 5.5 color classes are represented in angled brackets as labels associated to the places and, with a slight abuse of notation, to queuing stations.

The place `MaxTasks` represents the considered scheduling assignment

and contains tokens belonging to $< C >$ color class, initialized to $K_A$ and $K_B$ tokens for the corresponding task classes. Its marking represents the chosen configuration that allows a total of $K = K_A + K_B$ tasks simultaneously into a subsystem. The execution of a task starts with the firing of transition `Enter`, which can occur in one of the four following modes. When the subsystem is in normal operation, transition can fire in mode $1$ or $2$ removing respectively one token of class $A$ or $B$ from its input places, and creating customers of class $(A, A)$ or $(B, B)$ in the queue `Res1`, that represents the first resource of the subsystem. When there are no more tokens of either class $B$ or $A$ in place `Wait`, transition `Enter` fires in mode $3$ or $4$ allowing a task of class $A$ or $B$ to enter instead of the one that has already been completed. Queuing stations `Res1` and `Res2` represent the two resources of the subsystem. Even if the task classes generated by the CPN transition are $(A, A)$, $(A, B)$, $(B, B)$ and $(B, A)$, only the first component of each couple is used to determine the service requirements of a task. Instead, the second component is used in place `Rel` to allow transition `Leave` forwarding the correct task type to `Join` node, and to return the acquired task token into place `MaxTasks`. This is accomplished by the firing of transition `Leave` according to four modes, as summarized in Table **??**.

Each subsystem has a similar structure, and the same sub-model is repeated $n$ times as shown in Figure 5.5. Each subsystem can be characterized by different service demand for its resources, and different tasks allowances $K_A$ and $K_B$. When all the tasks have been completed, the `Join` primitive can fire, returning the customer to the reference station, and allowing the next job to start. To characterize the configuration of the system, the pool population mix and the subsystem population mix are denoted, respectively, as:

$$\begin{aligned} \vec{\alpha} &= \left( \alpha_A = \frac{N_A}{N_A + N_B}, \alpha_B = 1 - \alpha_A \right) \\ \vec{\beta} &= \left( \beta_A = \frac{K_A}{K_A + K_B}, \beta_B = 1 - \beta_A \right) \end{aligned} \tag{5.4}$$

Note that, while the scheduler generally has control over the subsystem population mix, $\vec{\beta}$, it cannot set the pool population mix, $\vec{\alpha}$. Indeed, the pool population mix depends on the type of application that is processed by the system, thus it changes when different applications are considered.

**Table 5.2:** *Transitions firing modes of the Petri net in the multi-formalism model.*

| **Transition** | **Mode** | **In$_1$** | **In$_2$** | **Out$_1$** | **Out$_2$** | **Description** |
|---|---|---|---|---|---|---|
| `Enter` | | `Wait` | `MaxTasks` | `Res1` | | |
| | 1 | $A$ | $A$ | $(A, A)$ | | Class A task |
| | 2 | $B$ | $B$ | $(B, B)$ | | Class B task |
| | 3 | $A$ | $B$ with `Wait`.$B = 0$ | $(A, B)$ | | Class A task, depletion |
| | 4 | $B$ | $A$ with `Wait`.$A = 0$ | $(B, A)$ | | Class B task, depletion |
| `Leave` | | `Rel` | | `MaxTasks` | `Join` | |
| | 1 | $(A, A)$ | | $A$ | $A$ | Class A task |
| | 2 | $(B, B)$ | | $B$ | $B$ | Class B task |
| | 3 | $(A, B)$ | | $B$ | $A$ | Class A task, depletion |
| | 4 | $(B, A)$ | | $A$ | $B$ | Class B task, depletion |

## 5.4 Results

### 5.4.1 Analytical results

Let us start considering the case of a Pool depletion system with a single subsystem. In order to derive the analytic equations its determine the depletion time, two assumptions (hereinafter referred to as *asymptotic assumptions*) have been made: *i)* the number of tasks initially in the pool must be much larger than the number of tasks simultaneously admitted in the subsystem, i.e., $N \gg K$; *ii)* the subsystem's capacity is large enough to saturate the subsystem itself.

When the two hypotheses are verified, the transient phases shown in Figure 5.2 are negligible and the depletion time is computed as $T = T_1 + T_2$, where $T_1$ and $T_2$ are the length of phases $\Phi_{\text{I}}$ and $\Phi_{\text{II}}$, respectively, without considering the imperceptible transient periods. In fact, when the asymptotic assumptions hold, the depletion time may be computed just accounting for the two dominant phases, $\Phi_{\text{I}}$ and $\Phi_{\text{II}}$.

While analyzing the asymptotic behavior of a Pool depletion system, phase $\Phi_{\text{I}}$ lasts until there is at least one task of each class in the system.

Being the pool and subsystem population mixes defined as in Eq. (5.4), the $\Phi_{\text{I}}$ length is given by:

$$T_1(N, \vec{\alpha}, \vec{\beta}) = min\left(\frac{N \cdot \alpha_A}{X_A(\beta_A)}, \frac{N \cdot \alpha_B}{X_B(\beta_A)}\right) \tag{5.5}$$

where $X_c(\beta_A)$ is the throughput for tasks of class $c \in \{A, B\}$ when there are $K_A = \beta_A K$ (thus, $K_B = (1 - \beta_A)K$) tasks in the subsystem. $\Phi_{\text{I}}$ ends when the pool runs out of tasks of one of the two classes.

During phase $\Phi_{\text{II}}$ the system always executes a single class workload (e.g., class $B$ in Figure 5.2). It starts with less than $N_c$ tasks – $c$ is the remaining class – since part of them have been already executed during $\Phi_{\text{I}}$ and it ends when there are no more tasks in the system. The length of phase $\Phi_{\text{II}}$, (i.e., $T_2$), is computed as:

$$T_2(N, \vec{\alpha}, \vec{\beta}) = \begin{cases} \frac{N\cdot\alpha_A - min\left(\frac{N\cdot\alpha_A}{X_A(\beta_A)}, \frac{N\cdot\alpha_B}{X_B(\beta_A)}\right) X_A(\beta_A)}{X_A(1)} \\ \frac{N\cdot\alpha_B - min\left(\frac{N\cdot\alpha_A}{X_A(\beta_A)}, \frac{N\cdot\alpha_B}{X_B(\beta_A)}\right) X_B(\beta_A)}{X_B(0)} \end{cases} \tag{5.6}$$

where $X_A(1)$ ($X_B(0)$) is the class $A$ ($B$) throughput when there are only tasks of class $A$ ($B$) in the system. Based on the class of the tasks in the pool when phase $\Phi_{\text{II}}$ begins, one of the two formulas in Eq. (5.6) is considered. For instance, if phase $\Phi_{\text{I}}$ ends when class $A$ tasks are exhausted (i.e.,

$\frac{N\cdot\alpha_A}{X_A(\beta_A)} < \frac{N\cdot\alpha_B}{X_B(\beta_A)}$), the second expression in Eq. (5.6) must be used, since the first one returns $0$. Instead, if the pool runs out of class B tasks when $\Phi_\mathrm{I}$ finishes, first formula in Eq. (5.6) is adopted.

To derive the asymptotic depletion time of the system, the definition previously presented is adopted and Eqs. (5.5) and (5.6) are summed as follows:

$$\begin{aligned} T(N,\vec{\alpha},\vec{\beta}) &= T_1(N,\vec{\alpha},\vec{\beta}) + T_2(N,\vec{\alpha},\vec{\beta}) \\ &= min\left(\frac{N\cdot\alpha_A}{X_A(\beta_A)}, \frac{N\cdot\alpha_B}{X_B(\beta_A)}\right) \\ &\quad + \frac{N\cdot\alpha_A - min\left(\frac{N\cdot\alpha_A}{X_A(\beta_A)}, \frac{N\cdot\alpha_B}{X_B(\beta_A)}\right) X_A(\beta_A)}{X_A(1)} \\ &\quad + \frac{N\cdot\alpha_B - min\left(\frac{N\cdot\alpha_A}{X_A(\beta_A)}, \frac{N\cdot\alpha_B}{X_B(\beta_A)}\right) X_B(\beta_A)}{X_B(0)} \end{aligned} \tag{5.7}$$

Both the formulas in Eq. (5.6) are used since, in any case, one of them is $0$ depending on the exhausted class.

After some algebraic manipulation, Eq. (5.7) becomes:

$$\begin{aligned} T(N,\vec{\alpha},\vec{\beta}) &= \frac{N\cdot\alpha_A}{X_A(1)} + \frac{N\cdot\alpha_B}{X_B(0)} \\ &\quad + min\left(\frac{N\cdot\alpha_A}{X_A(\beta_A)}, \frac{N\cdot\alpha_B}{X_B(\beta_A)}\right)\cdot\left(1 - \frac{X_A(\beta_A)}{X_A(1)} - \frac{X_B(\beta_A)}{X_B(0)}\right) \end{aligned} \tag{5.8}$$

Finally, due to the first asymptotic assumption (i.e., $N \gg K$), Eq. (5.8) is normalized over the initial number of tasks in the pool, $N$, without loss of generality:

$$\begin{aligned} \frac{T(N,\vec{\alpha},\vec{\beta})}{N} &= T(\vec{\alpha},\vec{\beta}) \\ &= \frac{\alpha_A}{X_A(1)} + \frac{1-\alpha_A}{X_B(0)} \\ &\quad + min\left(\frac{\alpha_A}{X_A(\beta_A)}, \frac{1-\alpha_A}{X_B(\beta_A)}\right)\left(1 - \frac{X_A(\beta_A)}{X_A(1)} - \frac{X_B(\beta_A)}{X_B(0)}\right) \end{aligned} \tag{5.9}$$

As said, all the equations presented in this section are for Pool depletion systems with only one subsystem. However, they may be easily adapted to the multi-subsystem case. In fact, the class $c \in \{A, B\}$ throughput of the single subsystem, $X_c(\beta_A)$, must be substituted by the sum of the class $c$ throughputs of the $S$ available subsystems, $\sum_{i=1}^{S} X_c^i(\beta_A)$.

Note also that, Eqs. (5.8) and (5.9) depend on the population mixes $\vec{\alpha}$ and $\vec{\beta}$. As previously said, the scheduler can easily control the population mix of the subsystem, while it is not usually allowed to modify the pool population mix. For this reason, two different minimum depletion times may be identified: the *relative* and the *absolute* ones. In the former case, the decision about the optimal operating point of the Pool depletion system analyzed is taken only varying the population mix $\vec{\beta}$. In the latter case, the scheduler is assumed to have control over both $\vec{\alpha}$ and $\vec{\beta}$. This topic is better analyzed in next section for the Pool depletion systems with one subsystem.

#### Relative minimum depletion time

Initially, the configuration analyzed consists of a system where all the parameters (i.e., $N_A$, $N_B$, $K$ and service demands matrix) are known, and only the fraction of tasks of the two classes in the subsystem (i.e., $K_A$ and $K_B$) varies. Referring to Eq. (5.9), the scheduler can vary only $\vec{\beta}$, whereas $\vec{\alpha}$ is given as an input parameter and is constant. In this situation, the scheduler must look for $\beta_A$ values such that the depletion time, computed through Eq. (5.9), is minimized.

Let us assume that the input parameters given by the user are the following service demands:

$$D_{rc} = \begin{bmatrix} 0.75 & 0.64 \\ 0.48 & 1.25 \end{bmatrix} \tag{5.10}$$

and $\alpha_A \in [0, 1]$. They are used to compute the normalized depletion time as in Eq. (5.9). The optimal population mix of the service demand matrix in Eq. (5.10) is derived through Eq. (2.2) and is $\vec{\beta}^* = (0.6, 0.4)$. Similarly, Eq. (2.4) can be used to determine the minimum and maximum values of the common saturation sector, that are $\vec{\beta}^{low} = (0.46, 0.54)$ and $\vec{\beta}^{up} = (0.73, 0.27)$, respectively.

The scheduler must analyze all the possible $\beta_A \in [0, 1]$ which may be adopted by the subsystem and identify the one that minimizes the depletion time.

Figure 5.6 shows the normalized depletion time, $T/N$, and the time required to complete phases $\Phi_{\mathrm{I}}$ and $\Phi_{\mathrm{II}}$, $T_1/N$ and $T_2/N$ respectively, when $\beta_A$ varies and for two different applications (two different values of $\alpha_A$ are considered: $\alpha_A = 0.4$ in Figure 5.6a and $\alpha_A = 0.8$ in Figure 5.6b).

In Figure 5.6a $T$ is minimum when $\beta_A \in [0.20, 0.73]$. The common saturation sector is included in the interval that minimizes the depletion time and both of them have the same upper bound. The minimum $T$ can

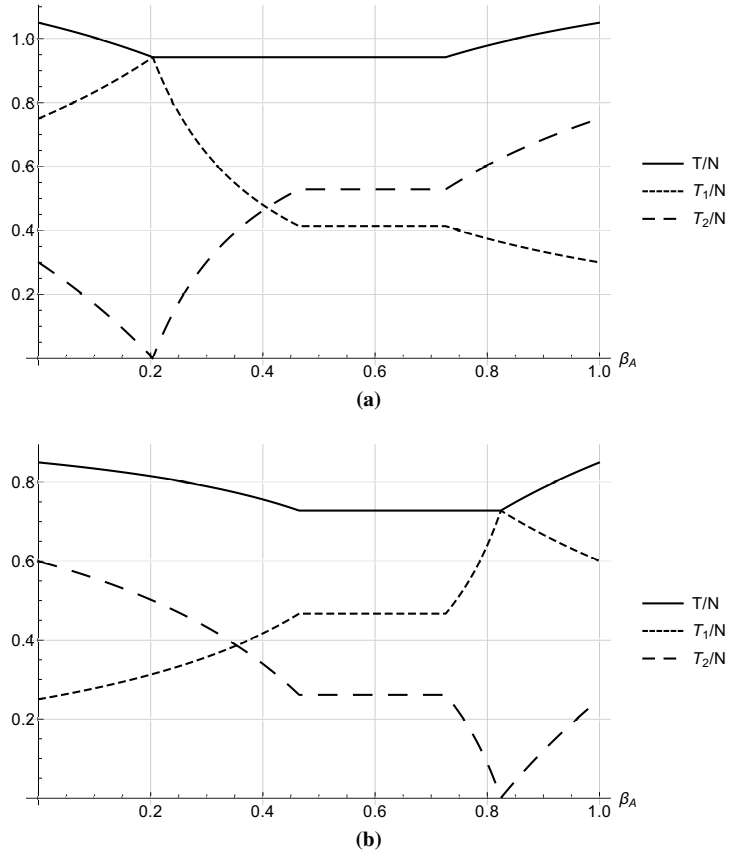



**Figure 5.6:** *Asymptotic depletion time as a function of parameter $\beta_A$ for (a) $\vec{\alpha} = (0.4, 0.6)$ and (b) $\vec{\alpha} = (0.8, 0.2)$.*

be identified looking for a $\vec{\beta}$ such that the execution of all class $A$ and class $B$ tasks ends at the same time. Let us denote such a value as $\vec{\beta}^+ = (K_A/K, K_B/K)$. If the system works with population mix $\vec{\beta}^+$, then:

$$N_A/X_A(\beta_A^+) = N_B/X_B(\beta_A^+).$$

Indeed, the system completes the execution of both the classes at the same time:

$$T_1(N, \vec{\alpha}, \vec{\beta}^+) = min\left(\frac{N_A}{X_A(\beta_A^+)}, \frac{N_B}{X_B(\beta_A^+)}\right) = \frac{N_A}{X_A(\beta_A^+)} = \frac{N_B}{X_B(\beta_A^+)}$$

and $T_2 = 0$.

Similar considerations can be drawn also for the results of Figure 5.6b. In that case, minimum depletion time is obtained for interval $[0.46, 0.82]$ that still includes the common saturation sector of the system considered.

It is interesting to note that when $\beta_A^+ < \beta_A^{low}$ (i.e., the lower bound of the common saturation sector) the depletion time is minimum for $\beta_A \in [\beta_A^+, \beta_A^{up}]$, whereas if $\beta_A^+ > \beta_A^{up}$ (i.e., the upper bound of the common saturation sector) the depletion time is minimum for $\beta_A \in [\beta_A^{low}, \beta_A^+]$. Instead, the depletion time is minimum for any $\beta_A \in \left[\beta_A^{low}, \beta_A^{up}\right]$ if $\beta_A^+$ belongs to the common saturation sector of the subsystem.

Recalling from [118] the formulas to compute the per class throughput of a system such as the one considered in this section (i.e., with $D_{1A} > D_{2A}$ and $D_{2B} > D_{1B}$):

$$X_A(\beta_A) = \begin{cases} \frac{\beta_A}{D_{2A}} & \beta_A < \beta_A^{low} \\ \frac{D_{2B}-D_{1B}}{D_{1A}D_{2B}-D_{2A}D_{1B}} & \beta_A^{low} \leq \beta_A \leq \beta_A^{up} \\ \frac{\beta_A}{D_{1A}} & \beta_A > \beta_A^{low} \end{cases}$$
$$X_B(\beta_A) = \begin{cases} \frac{1-\beta_A}{D_{2B}} & \beta_A < \beta_A^{low} \\ \frac{D_{2A}-D_{1A}}{D_{1B}D_{2A}-D_{1A}D_{2B}} & \beta_A^{low} \leq \beta_A \leq \beta_A^{up} \\ \frac{1-\beta_A}{D_{1B}} & \beta_A > \beta_A^{low} \end{cases} \tag{5.11}$$

it is possible to provide the equation for computing the value of $\beta_A^+$, when the asymptotic assumptions hold, imposing:

$$\frac{X_A(\beta_A)}{X_B(\beta_A)} = \frac{N_A}{N_B} \tag{5.12}$$

In fact, Algorithm 1 may be used to determine the $\beta_A^+$ value:

Line 1 derives the lower and upper bounds of the common saturation sector of the subsystem. Line 2 computes the value of $\beta_A^+$ assuming it is

**Algorithm 1** Algorithm to determine the value of $\beta_A^+$.

1: $\beta_A^{low} \leftarrow D_{2A}\frac{D_{2B}-D_{1B}}{D_{1A}D_{2B}-D_{2A}D_{1B}}$, $\beta_A^{up} \leftarrow D_{1A}\frac{D_{2B}-D_{1B}}{D_{1A}D_{2B}-D_{2A}D_{1B}}$
2: $\beta_A^+ \leftarrow \frac{N_A D_{2A}}{N_A D_{2A}+N_B D_{2B}}$
3: **if** $\beta_A^+ < \beta_A^{low}$ **then**
4: **return** $\beta_A^+$
5: **else**
6: $\beta_A^+ \leftarrow \frac{N_A D_{1A}}{N_A D_{1A}+N_B D_{1B}}$
7: **if** $\beta_A^+ > \beta_A^{up}$ **then**
8: **return** $\beta_A^+$
9: **else**
10: **return** $\beta_A^*$ {or any other value belonging to the common saturation sector}
11: **end if**
12: **end if**

lower than $\beta_A^{low}$, thus substituting the formulas for $\beta_A < \beta_A^{low}$ in Eq. (5.11) to Eq. (5.12). The hypothesis is verified at line 3: if it holds the $\beta_A^+$ derived is returned, otherwise it assumed to be larger than $\beta_A^{up}$ and the process is repeated at lines 6-8. If also the new hypothesis does not hold, it means class $A$ and class $B$ tasks are completed at the same time for any $\beta_A$ in the common saturation sector. For this reason, any value between $\beta_A^{low}$ and $\beta_A^{up}$ (e.g., the optimal one, $\beta_A^*$) may be returned.

Note that, also Algorithm 1 is valid only in the case considered (i.e., when $D_{1A} > D_{2A}$ and $D_{2B} > D_{1B}$). However, the formulas for the case where $D_{2A} > D_{1A}$ and $D_{1B} > D_{2B}$ are easily derivable since, as shown in Chapter 2, these conditions only affect the resource that saturates before and after the common saturation sector.

When the depletion time is minimum for several values of $\vec{\beta}$, the scheduler can also account for other performance metrics to make better choices about the operating point. For instance, it may consider the per-class response times. Indeed, it is able to make the system operate with the minimum depletion time and privilege one of the two classes making its tasks exit the system faster than the tasks of the other class.

In the case of a Pool depletion system with multiple subsystems, the optimal operating point $\vec{\beta}^*$ must be computed for each subsystem. Indeed, as shown in this section, the population mix of a subsystem depends only on the characteristics (i.e., the service demand matrix) of the subsystem itself.

Finally, it worths to notice that the minimum values of depletion time identified in this section are relative to the given values of $\vec{\alpha}$. A more general problem is studied in next section, where an absolute minimum is iden-

tified.

#### Absolute minimum depletion time

Let us now assume the scheduler is also able to modify $\vec{\alpha}$ (i.e., the population mix of the pool) in order to identify the shortest depletion time. Thus, considering the case of a single subsystem, the parameters passed by the users are the initial number of tasks $N$ in the pool, the capacity $K$ of the subsystem and the service demand matrix.

As said, the scheduler looks for the minimum depletion time varying both $\alpha_A$ and $\beta_A$ in Eq. (5.9). In particular, it operates according to the following steps: *i)* it computes the equi-utilization point $\beta_A^* \in [\beta_A^{low}, \beta_A^{up}]$ of the subsystem using Eq. (2.2); *ii)* it derives $X_A(\beta_A^*)$ and $X_B(\beta_A^*)$ through Eq. (5.11) using the formulas for $\beta_A^{low} \leq \beta_A \leq \beta_A^{up}$, since $\beta_A^* \in [\beta_A^{low}, \beta_A^{up}]$ as shown in Figure 2.1; *iii)* it computes the optimal pool population mix, $\vec{\alpha}^*$, substituting the throughputs previously computed into Eq. (5.12) and making some algebraic manipulations:

$$\begin{aligned} 1 + \tfrac{N_B}{N_A} &= 1 + \tfrac{X_B(\beta_A^*)}{X_A(\beta_A^*)} \\ \tfrac{N_A+N_B}{N_A} &= \tfrac{X_A(\beta_A^*)+X_B(\beta_A^*)}{X_A(\beta_A^*)} \\ \alpha_A^* &= \tfrac{X_A(\beta_A^*)}{X_A(\beta_A^*)+X_B(\beta_A^*)} \end{aligned} \tag{5.13}$$

*iv)* the scheduler makes the system work with $N_A/N = \alpha_A^*$ and with any number of tasks in the subsystem such that $K_A/K = \beta_A \in [\beta_A^{low}, \beta_A^{up}]$.

Figure 5.7 represents the depletion time of a Pool depletion system as a function of $\alpha_A$ and $\beta_A$, when the asymptotic assumptions hold and the subsystem is characterized by the service demand matrix in Eq. (5.10). In particular, it shows that an absolute minimum depletion time is found when $\alpha_A = \alpha_A^*$ and for any value of $\beta_A \in [\beta_A^{low}, \beta_A^{up}]$, that is, the common saturation sector of the subsystem. Note that, $\vec{\alpha}^*$ is the equi-load point of a system, is computed as shown in Eq. (2.3) and is $\vec{\alpha}^* = (0.69, 0.31)$ for the service demand matrix considered.

As previously seen, when the application is characterized by $\vec{\alpha} \neq \vec{\alpha}^*$, the system can only reach a relative minimum depletion time.

Note that, when considering a multi-subsystem Pool depletion system the optimal pool population mix is computed with an equation slightly different from Eq. (5.13). Indeed, the throughput per class of each subsystem

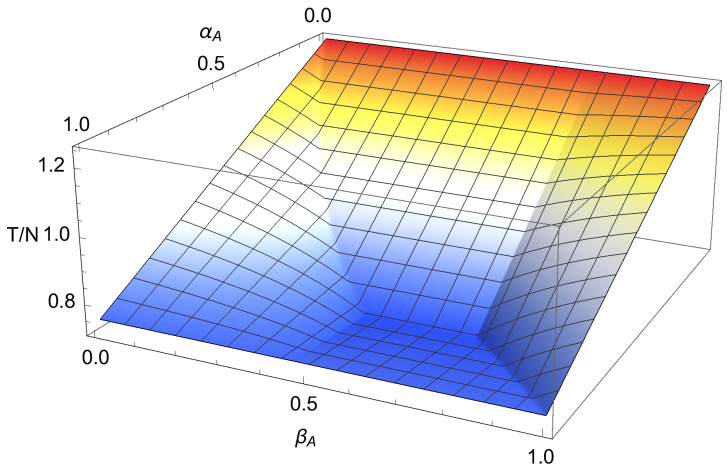



**Figure 5.7:** *Depletion time over the number of tasks initially into the pool as a function of $\alpha_A$ and $\beta_A$ when asymptotic assumptions hold.*

must be taken into account, thus:

$$\alpha_A^* = \frac{X_A(\beta_A^*)}{X_A(\beta_A^*) + X_B(\beta_A^*)} = \frac{\sum_{s=1}^{S} X_{A,s}(\beta_{A,s}^*)}{\sum_{s=1}^{S} X_{A,s}(\beta_{A,s}^*) + \sum_{s=1}^{S} X_{B,s}(\beta_{A,s}^*)} \tag{5.14}$$

Finally, the scheduling strategy that makes a Pool depletion system work with its optimal population mixes (i.e., $\vec{\beta}^*$ and, if possible, $\vec{\alpha}^*$) to minimize the depletion time of the applications it is executing is called *Optimal population mix* strategy.

Since this strategy operates on the execution order of the tasks generated by the application, it may be implemented together with the already available scheduling policies that have different targets (e.g., minimization of the execution time of each task, optimization of the system utilization) in order to further improve the system's performance.

### 5.4.2 Simulative results

When the asymptotic assumptions do not hold, the equations provided in Section 5.4.1 are no longer valid. For this reason, the Pool depletion system is simulated recurring to the multi-formalism model described in Section 5.3.2. The multi-formalism model has been validated against the CTMC model described in Section 5.3.1, but only Pool depletion systems with

small values of $N$ and $K$ and a single subsystem have been considered.

Indeed, the CTMC model is affected by the well known state space explosion problem and can provide the results in a reasonable time when $N \sim 100$ and $K \sim 10$. If more complex systems are considered, the state space explodes and the time required to complete the analysis is unacceptable.

In next sections, the multi-formalism model is used to study Pool depletion systems with single and multiple subsystems. In the case of multiple subsystems, a further distinction is considered: in fact, the subsystems may be identical – in this case we talk about homogeneous subsystems – or have different characteristics – also known as heterogeneous subsystems.

All the simulations provided in this chapter are performed with JSIMg, the JMT [11] simulator, and the results are computed with $99\%$ confidence interval.

#### Single subsystem

First of all, the non-asymptotic version of the *relative minimum* problem presented in Section 5.4.1 is studied for a Pool depletion system with only one subsystem. The service demand matrix used in this section is the one given in Eq. (5.10) and, as already said, the corresponding equi-load point is $\vec{\alpha}^* = (0.69, 0.31)$.

Since in the relative minimum problem the scheduler cannot change the pool population mix $\vec{\alpha}$, we assume that the application executed by the Pool depletion system has $\vec{\alpha} \simeq \vec{\alpha}^*$.

Figure 5.8 shows the length of the three phase when $N = 100$ and $K = 10$. The Pool depletion system has been analyzed for $\vec{N} = (70, 30)$, i.e., $\vec{\alpha} = (0.7, 0.3)$, whereas several values of $\vec{\beta}$ have been taken into consideration. Note that, the sum of the durations of all the phases depicted in Figure 5.8 is the depletion time.

Although $T_{\Phi 1}$ is maximum when $\beta_A = 0.6$, the minimum depletion time is observed when the system works with $\beta_A = 0.5$. That discrepancy is due to the non-asymptotic conditions of the system,. Indeed $N$ is not much larger than $K$, and $K$ is not large enough to saturate the subsystem. For these reasons, the transient periods (i.e., $T_{\tau 0}$, $T_{\tau 12}$ and $T_{\Phi 3}$) cannot be neglected and they affect the general system behavior.

Figure 5.9 depicts the amount of time saved when the Pool depletion system works with $\vec{\beta}$ different from the worst one (i.e., the subsystem population mix for which the maximum depletion time is observed). The per-

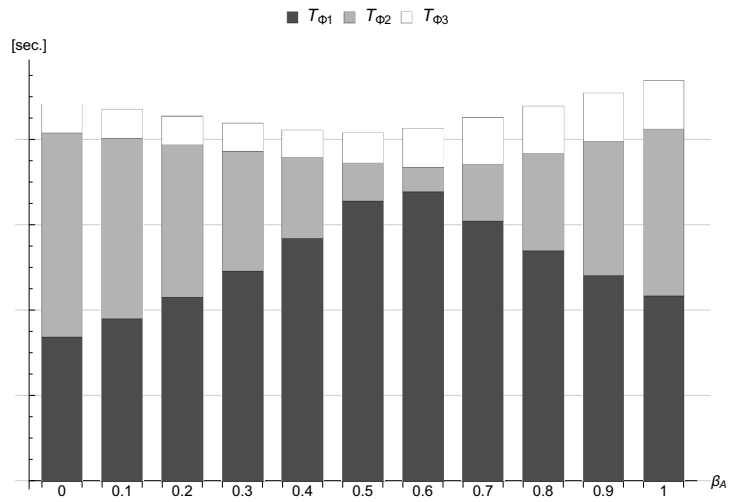



**Figure 5.8:** *Depletion time and length of each phase as a function of $\beta_A$, for the system with service demand matrix given in Eq. (5.10), when $\alpha_A = 0.7$, $N = 100$ and $K = 10$.*

centage of time saved is:

$$G = \frac{T_{max} - T_{estimated}}{T_{estimated}} \tag{5.15}$$

where $T_{max}$ is the maximum depletion time observed for the system analyzed and $T_{estimated}$ is the depletion time measured with the current $\vec{\beta}$.

Note that, depending on the service demand matrix considered, the gain obtained when the subsystem operates with its optimal population mix may be different, i.e., either larger or smaller.

Also the non-asymptotic behavior of the *absolute minimum* problem has been studied. For this purpose, the optimal population mix of the service demand matrix considered has been derived, i.e., $\vec{\beta}^* = (0.6, 0.4)$, and the system is assumed to work with that subsystem population mix. Thus, the depletion time is depicted against the $\alpha_A$ values, trying to identify the pool population mix for which the system executes, in the shortest time, all the tasks that are initially in the pool. The results are shown in Figure 5.10.

Differently from the asymptotic case, the Pool depletion system has its minimum depletion time for a value of $\vec{\alpha}$ different from the optimal one. In particular, the minimum $T$ is observed for $\alpha_A = 0.9$. Moreover, the maximum $T_{\Phi 1}$ of the system is not measured when $\vec{\alpha} = \vec{\alpha}^*$; however, it is observed for a $\alpha_A$ value (i.e., $\alpha_A = 0.65$) that is closer to the optimal one than the pool population mix for which the minimum $T$ is measured.

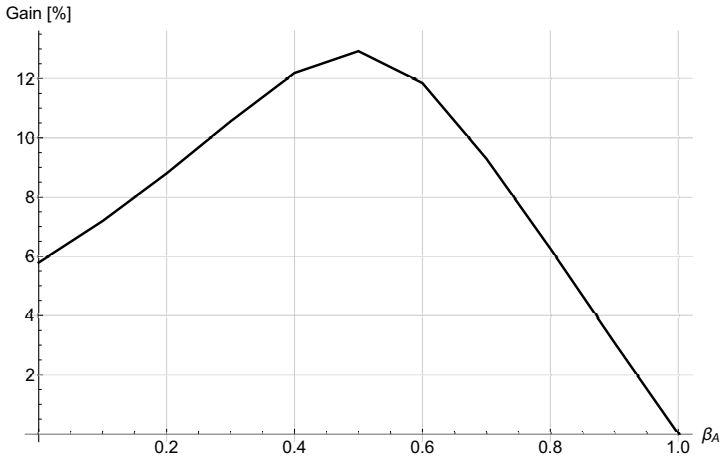



**Figure 5.9:** *Percentage of time saved when the system works with $\beta_A$ different from the one for which the system has the longest depletion time (i.e., $\beta_A = 1$).*

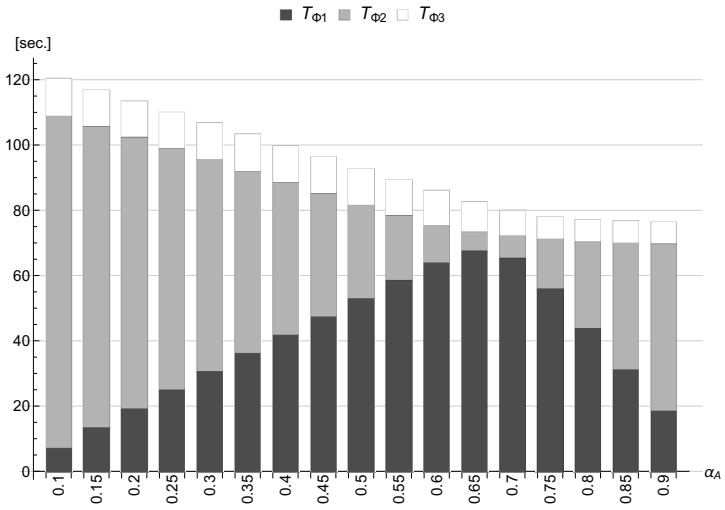



**Figure 5.10:** *Depletion time and length of each phase as a function of $\alpha_A$, for the system with service demand matrix given in Eq. (5.10), when $\beta_A = 0.6$, $N = 100$ and $K = 10$.*

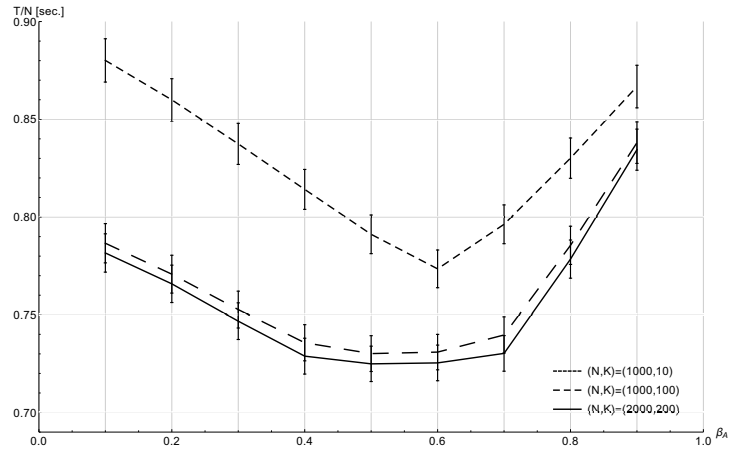



**Figure 5.11:** *Depletion time over number of tasks initially into the pool as a function of $\beta_A$, for different configurations of a Pool depletion system with $\alpha_A = 0.7$ and the service demand matrix given in Eq. (5.10).*

Finally, Figure 5.11 depicts the results obtained considering three different configurations of the system, that are $(N, K) = \{(1000, 10), (1000, 100), (2000, 200)\}$, assuming $\vec{\alpha} = (0.7, 0.3)$. The confidence intervals are shown in the graph.

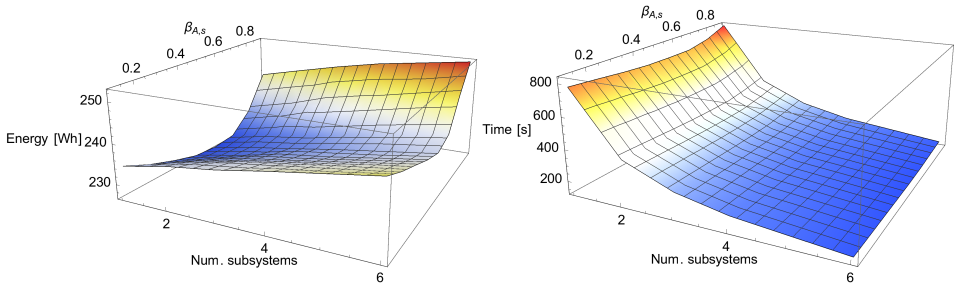
From those results we note that the larger is $K$, the flatter is the optimal region (i.e., the interval for which the depletion time is minimum). Indeed, if the subsystem capacity is larger, then it can saturate. This result supports the initial decision of considering asymptotic behavior of the system, and it seems reasonable to expect that the larger is the value of $K$, the lower will be the error done by the scheduler. Moreover, as long as the ratio of $K$ to $N$ is the same, the normalized depletion time does not seem to deeply vary.

#### Homogeneous subsystems

A Pool depletion system may be composed of identical (homogeneous) subsystems. The main advantage introduced by homogeneous distributed subsystems is the parallelization of jobs execution.

Since all the subsystems have the same characteristics, their optimal population mixes, $\vec{\beta}_s^*$ ($s$ is the subsystem considered), are identical. For this reason, all the subsystems have the same throughput and Eq. (5.14) coincides with Eq. (5.13).

In Figure 5.12 two performance metrics of a Pool depletion system with homogeneous distributed subsystems are depicted as a function of the num-




**(a)** **(b)**

**Figure 5.12:** *(a) Energy consumption and (b) depletion time of a Pool depletion system with homogeneous subsystem that works with* $\vec{\alpha} = \left(\frac{724}{1008}, \frac{284}{1008}\right)$*, as functions of total number of subsystems and their population mix.*

ber $S$ of available subsystems, and their population mix $\vec{\beta}$. The pool size of the considered system is $N = 1008$ and the capacity of each subsystem is $K = 108/S$. In the simulations, we consider $S = \{1, 2, 3, 4, 6\}$ and the service demand matrix presented in Eq. (5.10) is adopted to characterize all the subsystems of the Pool depletion system.

Since the service demand matrix does not change with respect to the one considered for the single subsystem case, the optimal population mix of each subsystem is $\vec{\beta}^* = (0.6, 0.4)$ and the equi-load point is $\vec{\alpha}^* = (0.69, 0.31)$. All the subsystems are identical, thus they all work with the same population mix.

For the sake of simplicity, only an application with $N_A = 724$ and $N_B = 284$, i.e., $\vec{\alpha} = \left(\frac{724}{1008}, \frac{284}{1008}\right)$, is considered.

Figure 5.12a represents the energy consumed to execute all the tasks initially in the pool when $P^1_{idle} = 315\ W$, $P^1_{busy} = 630\ W$, $P^2_{idle} = 250\ W$ and $P^2_{busy} = 500\ W$. $P^r_{idle}$ and $P^r_{busy}$ are the power consumptions of resource $r$ when it is idle and fully utilized, respectively.

Energy consumption has been derived through Eq. (2.6) after estimating the power consumption of each resource of the Pool depletion system using the linear power model described by Eq. (2.7). Note that, the depletion time is substituted to $T$ in Eq. (2.6).

In the configuration considered, the minimum value of energy consumption is observed when the system is working with only one subsystem. However, it is interesting to note that working with six subsystems and with the optimal population mix, $\vec{\beta}^*$, lets the service provider save more energy than working with only one subsystem and with a suboptimal population

mix.

Figure 5.12b shows the time required by the system to complete all the tasks initially in the pool (i.e., the depletion time). In this case, provisioning a large number of subsystems allows the service provider to parallelize the work. Thus, the depletion time can be reduced increasing the number of subsystems available for tasks execution.

Since the energy consumption and the depletion time have different optimal operating point, the system designer must take into account the service provider's priorities before configuring the Pool depletion system.

Finally, it worths to notice that for the application with $\vec{\alpha} = \left(\frac{724}{1008}, \frac{284}{1008}\right)$, both the metrics analyzed have their minimum point when all the available subsystems work with their optimal population mix, $\vec{\beta}^* = (0.6, 0.4)$.

#### Heterogeneous subsystems

The analysis of heterogeneous subsystems is interesting since it allows us to consider more general systems. For example, they can be used to model a data-center with different types of servers (e.g., new and old machines, fast and slow servers, etc.).

In order to study this type of environments, we focus on a Pool depletion system with two subsystems. Since they must have different features, the first subsystem is defined through the service demand matrix given in Eq. (5.10), whereas the following matrix defines the load of the second subsystem:

$$D_{rc} = \begin{bmatrix} 0.86 & 0.65 \\ 0.3 & 1.02 \end{bmatrix} \tag{5.16}$$

The optimal population mix of the service demand matrix in Eq. (5.16) is derived through Eq. (2.2) and represents the optimal population mix of the second subsystem, that is $\vec{\beta}_2^* = (0.3, 0.7)$.

As previously said, the optimal population mix of the pool must be computed with Eq. (5.14) if the system has heterogeneous subsystems, and in this case it is $\vec{\alpha}^* \simeq (0.55, 0.45)$. This result highlights another important aspect of Pool depletion systems with heterogeneous distributed subsystems, besides the parallelization of the workload. In fact, differently from the homogeneous case, using heterogeneous subsystems lets the service provider execute different applications with better performance. This result is shown in Figure 5.13, which depicts the depletion times of two different applications – that are characterized by two different pool population mixes, $\vec{\alpha}$ – executed by homogeneous and heterogeneous subsystems. The heterogeneous configuration provides a shorter depletion time than the homoge-

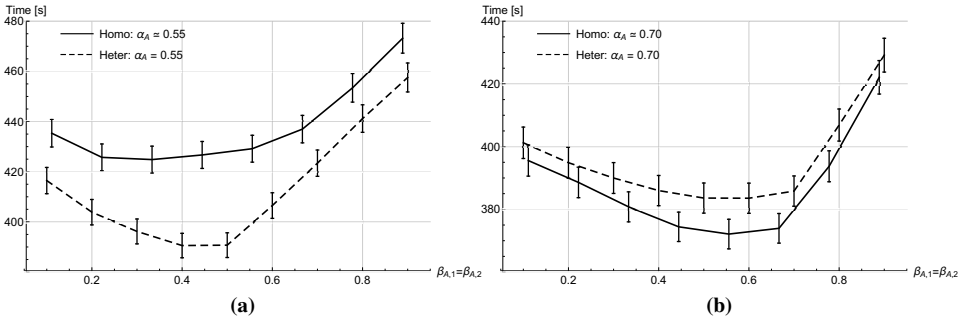



**Figure 5.13:** *Comparison of depletion time of Pool depletion systems with homogeneous and heterogeneous subsystems for two different applications: (a)* $\vec{\alpha} = (0.55, 0.45)$ *and (b)* $\vec{\alpha} = (0.70, 0.30)$.

neous one when it is serving application with $\vec{\alpha} = (0.55, 0.45) \simeq \vec{\alpha}^*_{heter}$ (i.e., the optimal pool population mix of the system with heterogeneous subsystems). On the contrary, if the application with $\vec{\alpha} = (0.70, 0.30)$ is considered, the homogeneous environment provides better performance than the heterogeneous one, since the new application is closer to the homogeneous optimal pool population mix, $\vec{\alpha}^*_{homo}$.

The energy consumption and depletion time of the system with two heterogeneous subsystems are depicted in Figure 5.14 for an application with $\vec{\alpha} \simeq \vec{\alpha}_{heter}$. Each metric is analyzed against the population mix of each subsystem, $\vec{\beta}_1$ and $\vec{\beta}_2$. In this case, for both the metrics, the minimum values are observed when each subsystem $s$ works with its optimal population mix, $\vec{\beta}^*_s$, i.e., $\vec{\beta}^*_1 = (0.6, 0.4)$ and $\vec{\beta}^*_2 = (0.3, 0.7)$, thus the scheduler can be easily configured to make any subsystem operate as near as possible to its optimal operating point.

### 5.4.3 Experimental results

The results obtained through analytic analysis and simulation are validated with experiments performed on PoliCloud [52], the private cloud designed and implemented by Politecnico di Milano, with 454 CPUs (1816 cores) and approximately 6 TB of storage.

In order to validate the results previously obtained, the one-subsystem case is considered. The pool is enabled by instantiating a virtual machine (VM) which starts a benchmark, generates the multi-class workload and monitors the whole system. A python script implements the tasks scheduling strategy.

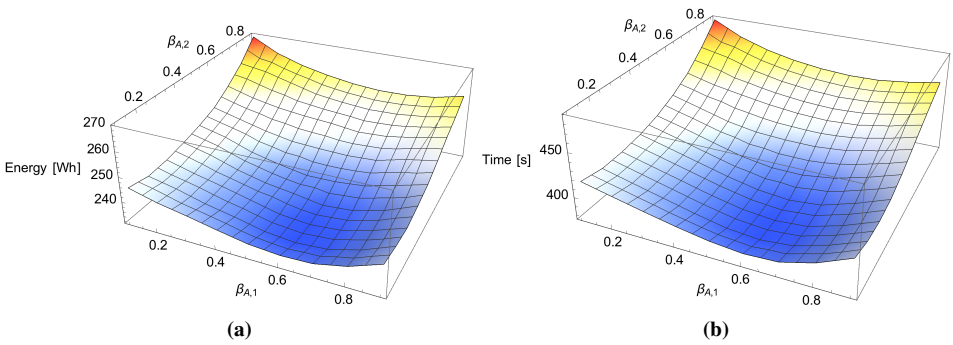



**Figure 5.14:** *(a) Energy consumption and (b) depletion time of a heterogeneous distributed system for* $\vec{\alpha} = (0.55, 0.45)$*, as functions of the population mix of each subsystem.*

The subsystem is composed by two VMs, that are the two resources modeled in the Pool depletion system framework. Both the VMs into the subsystem are identically instantiated with one CPU, 2 GB memory and the Ubuntu Server 16.04 operating system.

The tasks composing the workload are generated using *sysbench* [81], a modular, cross-platform and multi-threaded benchmark tool. Each task is CPU-bound and requires the computation of a certain value of prime numbers; such a value is passed as input parameter to the tool. Note that, the amount of prime numbers generated and the time required to complete the task are linearly related as shown in Figure 5.15.

Specifying a different amount of prime numbers to be generated by *sysbench*, it is possible to take into consideration a multi-class workload. In fact, two possibly different values representing the prime numbers to be generated (i.e., $x_A$ for class $A$ and $x_B$ for class $B$) are passed to each VM before starting the benchmark. Due to the linear relationship existing between the amount of prime numbers to be generated and the time to complete each task, if the former is exponentially distributed, also the latter follows the same distribution. Thus, whenever a class $c$ task is sent to a VM, the amount of prime numbers to be generated by that task follows an exponential distribution with average $x_c$.

The total time each task spends into a VM also accounts for the other operations performed by the physical machines hosting the VMs and the connection time between the VMs involved.

For the purpose of these experiments, the average service demands con-

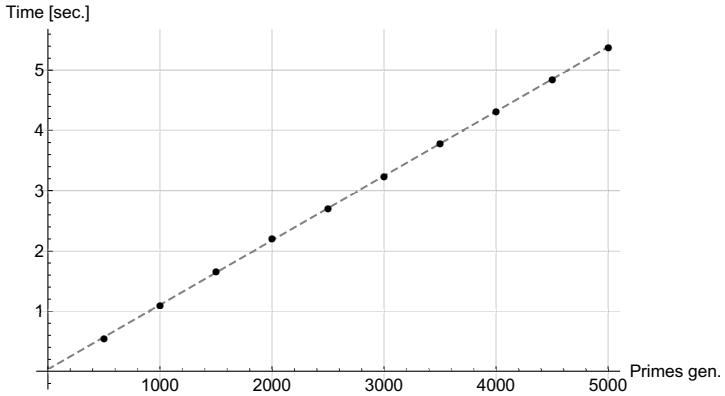



**Figure 5.15:** *Q-Q plot that compares the number of primes generated and the time required to complete that operation, showing the linear relationship existing between the two quantities.*

sidered are given by the following matrix (in seconds):

$$D_{rc} = \begin{bmatrix} 4.30889 & 3.48539 \\ 2.39562 & 6.57502 \end{bmatrix} \tag{5.17}$$

for which $\vec{\alpha}^* = (0.62, 0.38)$ and $\vec{\beta}^* = (0.52, 0.48)$. The pool size is set to $N = 100$ with $\vec{\alpha} = (0.6, 0.4) \simeq \vec{\alpha}^*$, whereas the subsystem capacity is $K = 10$ and its population mix $\beta_A$ varies between 0 and 1.

The average amount of prime numbers to be generated to make each VM work with the mean service demands given by Eq. (5.17) are reported in the following matrix:

$$\text{NumPrimes}_{rc} = \begin{bmatrix} 3497 & 2773 \\ 1744 & 5141 \end{bmatrix} \tag{5.18}$$

Each experiment is repeated 100 times and the average depletion time for each configuration considered is depicted in Figure 5.16.

The first and second bars are the depletion time observed in the real cloud environment and estimated by the multi-formalism model, respectively. They refer to the left y-axis and their $99\%$ confidence intervals are also depicted. The minimum depletion time is observed for $\vec{\beta} = (0.5, 0.5) \simeq \vec{\beta}^*$. When this Pool depletion system works with its optimal operating point, it can save up to $15\%$ of time required to complete the application when a different population mix is adopted.

The third bar represents the Mean absolute percentage error (MAPE)

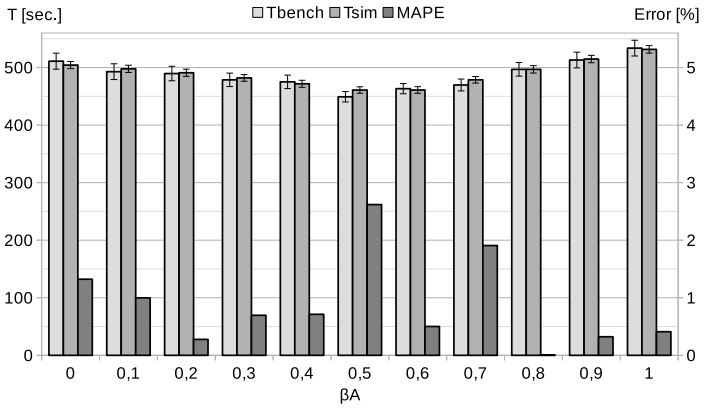



**Figure 5.16:** *Depletion time of a Pool depletion system observed in a real cloud environment (first bar) and estimated through the multi-formalism model proposed in Section 5.3.2 (second bar). The MAPE of the model is shown by the third bar that refers to the right y-axis.*

computed as:

$$\text{MAPE} = \frac{|T_{bench} - T_{sim}|}{T_{bench}} \tag{5.19}$$

where $T_{bench}$ is the average depletion time observed during the experiment and $T_{sim}$ is the estimated one. MAPE refers to the right y-axis and shows that the largest error made by the model, that is for $\vec{\beta} = (0.5, 0.5)$, is lower than $3\%$.

That proves the multi-formalism model proposed in this chapter can estimate the depletion time of a Pool depletion system with very good accuracy.

## 5.5 Exploitation

Two real applications are now considered to show some possible exploitations of Pool depletion systems. In the first case, an Apache Hive based application is analyzed, whereas in the second example, a battery-operated Pool depletion system is studied.

### 5.5.1 Apache Hive

Although Pool depletion systems can model many different Big Data applications, the case-study we consider in this section is an Apache Hive [132] based application. Apache Hive is an open source data warehouse system and is used on top of Apache Hadoop. It has been extensively adopted by

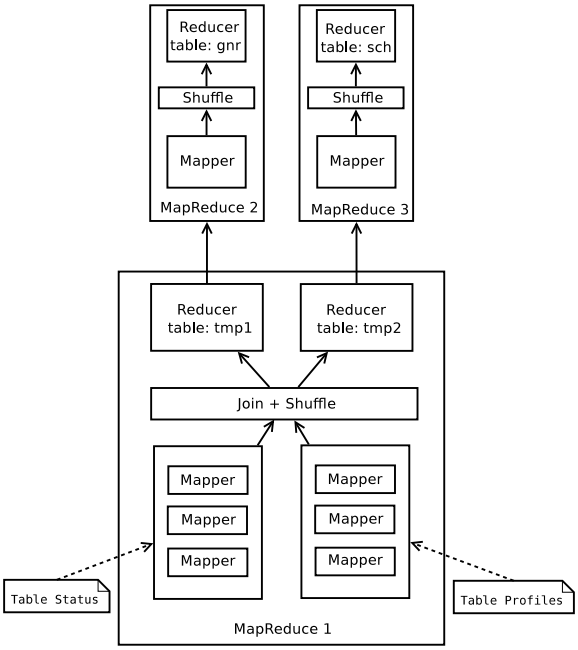



**Figure 5.17:** *Query plan of three MapReduce jobs analyzed in [132].*

many organizations (e.g., Facebook, Netflix, Spotify) since it makes easier the management of large data and their queries. Hive was introduced by Facebook in [132]. It may be run on different Hadoop's frameworks (e.g., MapReduce, YARN, Spark, Tez) and provides a SQL-like query language that is called HiveQL.

A query to retrieve most popular Facebook status based on users' gender and school is investigated in [132]. Its simplified query plan is shown in Figure 5.17, where three MapReduce jobs are represented.

To study the performance of *Optimal population mix* strategy introduced in Section 5.4, the depletion time of a system adopting that policy is compared with the depletion time of the same system when all the class $A$ tasks are executed before the class $B$ ones, and vice-versa. In particular, we focus on the *MapReduce 1* join function of the Hive query in Figure 5.17.

In that case, both Map and Reduce have multi-class workloads. Indeed, Map phase gets its data from two different tables through *retrieving* tasks (i.e., I/O bound) and must execute *joining* tasks (i.e., CPU bound) in order to join the data retrieved; instead, the Reduce phase must still process the data and then write two temporary tables.

Assuming two tables must be joint, we wish to identify the best way to execute all the tasks of a MapReduce job in order to decrease the total time

required to complete the join clause and get the expected results.

The Map and Reduce phases are modeled by two different Pool depletion systems with heterogeneous subsystems, and have been characterized starting from the parameters derived in [148] and [6]. The former provides some data about MapReduce jobs executed on Facebook's clusters, the latter refers to a public Hadoop repository[6]. Thus, the Map system is defined by the following service demands (in seconds):

$$D_{rc,1}^{Map} = \begin{bmatrix} 25 & 21 \\ 16 & 39 \end{bmatrix} \qquad D_{rc,2}^{Map} = \begin{bmatrix} 29 & 22 \\ 10 & 34 \end{bmatrix} \tag{5.20}$$

whereas the Reduce one has the following service demands (still in seconds):

$$D_{rc,1}^{Red} = \begin{bmatrix} 55 & 39 \\ 28 & 82 \end{bmatrix} \qquad D_{rc,2}^{Red} = \begin{bmatrix} 60 & 46 \\ 17 & 70 \end{bmatrix} \tag{5.21}$$

Let us assume the MapReduce job is splitted into $N^{Map} = 1000$ and $N^{Red} = 200$ tasks before Map and Reduce phases start, respectively. The number of tasks that is concurrently executed during the two phases is $K^{Map} = 100$ for Map and $K^{Red} = 50$ for Reduce. The optimal population mix of each subsystem, derived using Eq. (2.2), are $\vec{\beta}_1^{*,Map} = (0.58, 0.42)$, $\vec{\beta}_2^{*,Map} = (0.29, 0.71)$, $\vec{\beta}_1^{*,Red} = (0.52, 0.48)$ and $\vec{\beta}_2^{*,Red} = (0.25, 0.75)$.

Finally, an application whose $\vec{\alpha}^{Map} = (0.53, 0.47)$ for Map phase and $\vec{\alpha}^{Red} = (0.49, 0.51)$ for the Reduce one has been considered. For both Map and Reduce phases $\vec{\alpha} \simeq \vec{\alpha}^*$, where the equi-load point of each phase has been computed through Eq. (5.14).

The results of the simulations have been computed with $99\%$ confidence intervals, and they are shown in Figure 5.18. As said, besides the *Optimal population mix* strategy, also the *First A Then B* and *First B Then A* strategies have been observed to investigate the improvement enabled by our scheduling strategy.

In Figure 5.18 the time to complete all the Map and Reduce tasks into the system are compared based on the scheduling policy adopted. The total time to complete a MapReduce job is also depicted. For the sake of simplicity, we assume Reduce tasks are served after the Map ones and no parallelism between tasks belonging to different phases is admitted (i.e., *Total length = Map phase length + Reduce phase length*). Due to the large number of Map tasks that are generated, this phase is affecting the

[6]Available at `http://ftp.pdl.cmu.edu/pub/datasets/hla/`. Accessed: Jan. 15, 2018. Please, include *http* at the beginning of the URL.

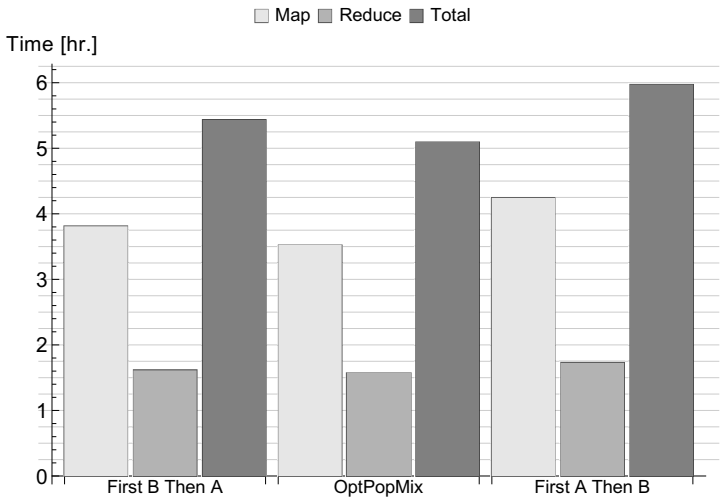



**Figure 5.18:** *Time to complete the MapReduce 1 job adopting different scheduling strategies.*

global performance of the system more than the Reduce one. When the subsystems work with their optimal population mixes, the service provider complete all the tasks (i.e., the *MapReduce 1* job) in a shorter time than using other strategies. The system requires the longest time to complete the MapReduce job when it executes the *class B* tasks after completing all the *class A* ones.

Based on the observed results, *Optimal population mix* strategy lets the system save from $5\%$ to $15\%$ of the total time, with respect to other strategies. As said, this result may be further increased; in fact, *Optimal population mix* strategy can be implemented into a system together with other scheduling policies that operates at different levels and with different targets.

### 5.5.2 Battery-operated Pool depletion systems

The analysis of a Pool depletion system may be also exploited in order to compute the probability of completing all the tasks in the pool before the battery that keeps the whole system on runs out of power. For instance, let us assume the main power source fails as soon as a new job (e.g., a Big Data application) enters a Pool depletion system to be processed. Thus, the uninterruptible power supply (UPS) starts working to keep the server on.

The goal of this analysis is to investigate if all the tasks initially in the pool can be executed before the UPS battery is completely discharged and

the Pool depletion system is shut down. The results obtained from this study may be used to configure the UPS in order to make it suitable for the service provider's purposes.

Several battery models have been proposed in literature in order to investigate the life of a battery: some of them, such as the electro-chemical models [44], are very accurate and need several input parameters to work; some others, like the kinetic battery model (KiBaM) [71, 77], need just a few parameters and may be easily used also by inexperienced users. In this section, the latter model is adopted.

The KiBaM is an accurate and simple model [71, 77] that divides the stored charge into two different wells: the *available charge* and the *bound charge*. The two wells are connected by a pipe and the charge can migrate in both directions – based on the level of charge in the two wells – at a given rate $\omega$. At the beginning, a fraction $\phi = [0, 1]$ of the total capacity is in the available charge well, and the remaining $1 - \phi$ fraction is in the bound charge well. When the battery is strained, only the available charge may be used, whereas the bound charge slowly becomes available.

The battery lifetime $L$ is derived in [77] by solving the differential equations that models the change of the charge of both the wells and, when the load is constant, it may be computed determining the variable $t$ in the equation:

$$L = t \mid I \cdot t + \frac{(1-\phi) \cdot I}{\phi} \cdot \frac{1 - e^{-\omega' t}}{\omega'} = \hat{\gamma} \tag{5.22}$$

where $I = P(U)/\Delta V$ is the load current that strains the battery (where $P(U)$ is the Pool depletion system power consumption as a function of utilization $U$ and $\Delta V$ is the voltage of the battery), $t$ is the lifetime of the battery, $\hat{\gamma}$ is the battery full capacity and $\omega' = \omega/[\phi(1-\phi)]$. In this section, $\phi = 0.5$ and $\omega = 0.01$ [71].

Let us consider a Pool depletion system with $N = 1000$ tasks initially in the pool and one subsystem composed by a CPU and a disk, whose capacity is $K = 100$ and with the following service demand matrix:

$$D_{rc} = \begin{bmatrix} 3.84 & 2.3808 \\ 0.96 & 6 \end{bmatrix} \tag{5.23}$$

whose equi-utilization and equi-load points are $\vec{\beta}^* = (0.4, 0.6)$ and $\vec{\alpha}^* = (0.56, 0.44)$, respectively.

The Pool depletion system power consumption is computed recurring to the linear power model in Eq. (2.7) and assuming $P_{idle}^{cpu} = 60W$, $P_{busy}^{cpu} = 100W$, $P_{idle}^{disk} = 0W$ and $P_{busy}^{disk} = 20W$. The UPS battery parameters

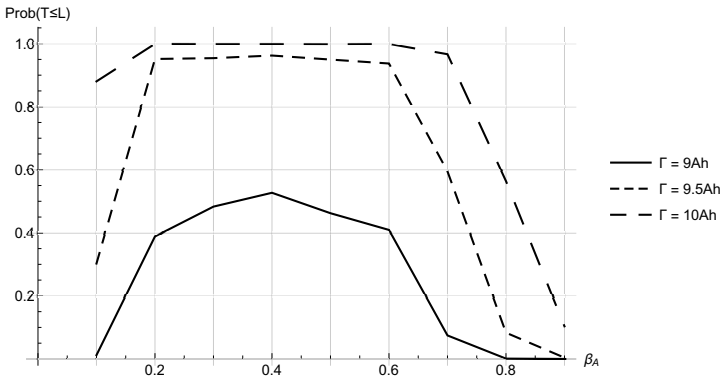



**Figure 5.19:** *Probability that all the $N$ tasks initially in the pool are served before the UPS battery runs out of power, $P(T \leq L)$, as a function of $\beta_A$, when the pool population mix is set to $\vec{\alpha} = \vec{\alpha^*}$ and different battery capacity $\hat{\gamma}$ are considered.*

are obtained from Trust OXXTRON 1000VA UPS[7] specifications and are $\Delta V = 12$ volt and $\hat{\gamma} = 9$Ah.

The Pool depletion system described is simulated with JSIMg [11] when its pool population mix is $\vec{\alpha} = \vec{\alpha^*} = (0.56, 0.44)$. The cumulative distribution function of the depletion time for many values of $\vec{\beta}$ has been evaluated. Thus, the probability of processing all the $N$ tasks before the battery is exhausted, i.e., $Prob(T \leq L)$, is derived and the results are plotted in Figure 5.19.

When a battery with $9$Ah capacity is used, the probability to process all the tasks initially in the pool before the battery completely drains is only the $50\%$ if the system implements the *Optimal population mix* strategy. That probability is lower if the subsystem is not working with its $\vec{\beta^*}$. For this reason, the system provider should consider a different UPS. In particular, a battery with a larger initial capacity is required. For instance, increasing the battery capacity of $1$Ah allows the Pool depletion system to complete all the tasks before the battery is empty for several subsystem's population mixes.

[7] `http://www.trust.com/en/product/17680-oxxtron-1000va-ups`. Accessed: Jan. 15, 2018.

# Part III

# Epistemic uncertainty propagation

CHAPTER 6

# Parametric Uncertainty Propagation in M/M/1 Queue

This chapter analyzes epistemic uncertainty propagation in M/M/1 queues, proposing two different approaches to evaluate how this kind of uncertainty affects the output measure of the model considered. The M/M/1 queuing model has been taken into account due to its relevance in the rest of this thesis. Indeed, M/M/1 queues are are the basic elements of both *Pool depletion systems* (Chapter 5) and *Mobile CrowdSensing* (Chapter 9) models.

This chapter is based on research conducted with *Universitá degli Studi dell'Aquila* and *Duke University*. In fact, we mainly focused on the Erlang case described in Section 6.2.2, whereas the colleagues from *Universitá degli Studi dell'Aquila* primarily worked on the Uniform case introduced in Section 6.2.1.

Although we mainly focused on Erlang distributed input parameters, the results about the Uniform case are also presented in this thesis to provide comparison with a different approach to the same problem. In this way, a broader view about epistemic uncertainty propagation is given. Thus, the purpose of assuming Uniform distribution for each input parameter is not to introduce non-determinism. In fact, Markov Decision Processes may be

used to study the effect of uncertainty on non-deterministic models [121, 122].

## 6.1 Motivation

The well known M/M/1 queue is revisited in this chapter considering the epistemic (parametric) uncertainty. Two input parameters, namely the arrival rate $\lambda$ and the service rate $\mu$ of the model are generally considered as fixed values while solving the model. In practice, these parameters are estimated from measurements, thus they are given in the form of confidence intervals, in addition to their point estimates. It behooves the analyst to consider the effects that the confidence intervals on the input parameters have on the output results of the queuing model. The objective of this chapter is to show how to carry out such parametric uncertainty propagation to derive confidence intervals on the steady state outputs of the M/M/1 queue, such as system response time $R$ and number of customers being processed by the queue $N$.

In carrying out such uncertainty propagation, the arrival and the service rates are seen to be random variables themselves. As they vary over their respective support, not only the stability condition needs to be maintained, but also computational instability arises when the values of the two parameters are nearly equal. This makes epistemic uncertainty propagation somewhat delicate. Two different approaches to propagate epistemic uncertainty are investigated and shown, thus to provide confidence intervals on the two output measures of the M/M/1 queue (i.e., $R$ and $N$), given confidence intervals of the input parameters.

The approaches described in this chapter allow to take early decisions about the system design under two different hypotheses about the distributions of input parameters. For example, assume to have some requirements on the response time. With the techniques discussed, the risk of violating that requirement under the current uncertainty in the system and environment parameters may be studied. In fact, the probability that response time is shorter than the required value may be calculated. This evaluation may lead to two types of considerations about the system design: first, do not place the system in an environment where the arrival rate is larger than a threshold $\bar{\lambda}$; second, make design decisions that lead the service rate to be higher than another threshold $\bar{\mu}$.

## 6.2 Basic equations for uncertainty propagation

This section provides the mathematical calculations for analyzing an M/M/1 queues with a certain degree of uncertainty on their arrival and service rates. In particular, it is known by M/M/1 definition that inter-arrival and service times are exponentially distributed, but their rates (or mean values) are assumed to be uncertain.

M/M/1 queues are usually adopted to analyze queuing models [94]. Some examples of current applications that have been modeled with exponential distributed arrival and service times are data-centers and cloud resources [16, 104, 143].

Let us introduce a probability space $(\Omega, \mathcal{F}, P)$, where $\Lambda$ and $M$ are two random variables respectively defined as the arrival and the service rates of an M/M/1 system. We assume these two random variables to be independent, hence their joint probability density can be written as: $f_{\Lambda,M}(\lambda, \mu) = f_{\Lambda}(\lambda) f_{M}(\mu)$.

In this setting, if the average number of entities in the system is denoted by $N$, it is itself a random variable depending on the pair $(\Lambda, M)$. That is to say that $N$ is known only conditionally to $(\Lambda, M)$:

$$N|_{\Lambda=\lambda, M=\mu} = \frac{\lambda}{\mu - \lambda}.$$

In a similar way, denoting by $R$ the average response time of the system, that is again a random variable depending on the same pair $(\Lambda, M)$ as follows:

$$R|_{\Lambda=\lambda, M=\mu} = \frac{1}{\mu - \lambda}.$$

The system must be studied under stability condition, namely when the arrival rate is lower than the service one, so that the average number of entities and the average response time do not tend to infinity. Hence, in order to limit the divergence of $N$ and $R$ when $\lambda$ and $\mu$ are very close, two different limitations on the random variables joint region are introduced and they are defined as follows:

$$B_k = \{(\lambda, \mu) : \ \mu > k\lambda\}, \qquad \text{for some } k > 1$$

or

$$B_\epsilon = \{(\lambda, \mu) : \ \mu > \lambda \ + \epsilon\}, \qquad \text{for some } \epsilon > 0.$$

The condition $\mu > k\lambda$ is equivalent to the condition

$$N|_{\Lambda=\lambda, M=\mu} < \frac{1}{k-1},$$

whereas $\mu > \lambda \ + \epsilon$ is equivalent to

$$R|_{\Lambda=\lambda, M=\mu} < \frac{1}{\epsilon}.$$

Furthermore, both regions are independent of the values of $\Lambda$ and $M$, and the two events are denoted as:

$$A_k = \{(\Lambda, M) \in B_k\} = \left\{ N < \frac{1}{k-1} \right\}$$

and

$$A_\epsilon = \{(\Lambda, M) \in B_\epsilon\} = \left\{ R < \frac{1}{\epsilon} \right\}.$$

Taking this into account, it is possible to compute:

$$\mathbb{E}[N|A_k] = \frac{1}{P(A_k)}\mathbb{E}[N\mathbf{1}_{A_k}] \quad \text{or} \quad \mathbb{E}[N|A_\epsilon] = \frac{1}{P(A_\epsilon)}\mathbb{E}[N\mathbf{1}_{A_\epsilon}],$$

and

$$\mathbb{E}[R|A_k] = \frac{1}{P(A_k)}\mathbb{E}[R\mathbf{1}_{A_k}] \quad \text{or} \quad \mathbb{E}[R|A_\epsilon] = \frac{1}{P(A_\epsilon)}\mathbb{E}[R\mathbf{1}_{A_\epsilon}],$$

which reduces to derive the numerators and denominators separately.

To simplify the presentation, only the average number of entities in the system is considered in the following. For the denominators, we have:

$$\begin{aligned} P(A_k) &= P((\Lambda, M) \in B_k) = \iint_{B_k} f_{\Lambda,M}(\lambda, \mu) d\lambda d\mu \\ P(A_\epsilon) &= P((\Lambda, M) \in B_\epsilon) = \iint_{B_\epsilon} f_{\Lambda,M}(\lambda, \mu) d\lambda d\mu \end{aligned} \tag{6.1}$$

whereas the numerators are derived using the theorem of total expectation [133]:

$$\begin{aligned} \mathbb{E}[N\mathbf{1}_{A_k}] &= \mathbb{E}\left[\frac{\Lambda}{M-\Lambda}\mathbf{1}_{A_k}\right] = \mathbb{E}\left[\frac{\Lambda}{M-\Lambda}\mathbf{1}_{B_k}(\Lambda, M)\right] \\ &= \int_{\mathbb{R}}\int_{\mathbb{R}} \frac{\lambda}{\mu-\lambda}\mathbf{1}_{B_k}(\lambda, \mu) f_{\Lambda,M}(\lambda, \mu) d\lambda d\mu \\ &= \int\int_{B_k} \frac{\lambda}{\mu-\lambda} f_{\Lambda,M}(\lambda, \mu) d\lambda d\mu \end{aligned} \tag{6.2}$$

where the second equality is justified by the random variable $\mathbf{1}_{A_k}$ that is measurable with respect to the $\sigma-$algebra generated by $(\Lambda, M)$.

The variance of the same random variable is computed as:

$$\begin{aligned} Var[N|A_k] &= \mathbb{E}[N^2|A_k] - (\mathbb{E}[N|A_k])^2 \\ &= \frac{1}{P(A_k)}\mathbb{E}[N^2\mathbf{1}_{A_k}] - (\mathbb{E}[N|A_k])^2 \end{aligned} \tag{6.3}$$

therefore reducing the problem to derive:

$$\mathbb{E}[N^2\mathbf{1}_{A_k}] = \mathbb{E}[\mathbb{E}[N^2\mathbf{1}_{A_k}|\Lambda, M]] = \mathbb{E}\left[\frac{\Lambda^2}{(M-\Lambda)^2}\mathbf{1}_{\{M>k\Lambda\}}\right] \tag{6.4}$$

The expected number of entities into the system, when we restrict to the second type of region, is formalized as:

$$\begin{aligned} \mathbb{E}[N\mathbf{1}_{A_\epsilon}] &= \mathbb{E}\left[\frac{\Lambda}{M-\Lambda}\mathbf{1}_{A_\epsilon}\right] = \mathbb{E}\left[\frac{\Lambda}{M-\Lambda}\mathbf{1}_{B_\epsilon}(\Lambda, M)\right] \\ &= \int\int_{B_\epsilon} \frac{\lambda}{\mu-\lambda} f_{\Lambda,M}(\lambda,\mu)d\lambda d\mu \end{aligned} \tag{6.5}$$

and the variance expression is written accordingly. Analogous formulas can be written also for the response time random variable $R$.

As shown in the subsequent sections, there are interesting cases for the probability densities $f_\Lambda(\lambda)$ and $f_M(\mu)$ for which computations are quite easy for the outputs such as: $\mathbb{E}[N\mathbf{1}_{A_k}]$, $\mathbb{E}[N\mathbf{1}_{A_\epsilon}]$, $\mathbb{E}[R\mathbf{1}_{A_k}]$, $\mathbb{E}[R\mathbf{1}_{A_\epsilon}]$. To take into account epistemic uncertainty in the M/M/1 input parameters, two different distributions for the densities are considered: Uniform and Erlang. The former distribution corresponds to the case when one has no information regarding the behavior of the rates within known bounds, hence considering uniform density seems to be the most natural assumption one can make. The latter distribution is inspired by the inter-arrival and service times that are exponentially distributed. In this case, as shown and proved in [98, 125, 133], $\Lambda$ follows an $r$-stage Erlang distribution with rate $s$, and $M$ has a $n$-stage Erlang distribution with rate $u$. Parameters $s$ and $u$ are defined respectively as the sum of $r$ samples collected for inter-arrival times, and as the sum of the $n$ samples collected for service times.

### 6.2.1 The Uniform case

In this case, no specific knowledge of the random variables $\Lambda$ and $M$ are available, but we assume that they are both uniformly distributed on some known lower (L) and upper (U) bound intervals, that are denoted respectively by $[\lambda_L, \lambda_U]$ and $[\mu_L, \mu_U]$. Thus, their joint probability density is given

by:

$$\begin{aligned} f_{\Lambda,M}(\lambda,\mu) &= f_{\Lambda}(\lambda) f_{M}(\mu) \\ &= \frac{1}{(\mu_U-\mu_L)(\lambda_U-\lambda_L)} \mathbf{1}_{[\lambda_L,\lambda_U]\times[\mu_L,\mu_U]}(\lambda,\mu) \\ &=: \frac{1}{C}\mathbf{1}_B(\lambda,\mu) \end{aligned} \tag{6.6}$$

with $C = (\lambda_U - \lambda_L) \cdot (\mu_U - \mu_L)$ that denotes the area of rectangle $B = [\lambda_L, \lambda_U] \times [\mu_L, \mu_U]$ where the joint probability density is concentrated.

Thus, substituting Eq. (6.6) into Eq. (6.1):

$$P(A_k) = \frac{C_k}{C} \quad \text{and} \quad P(A_\epsilon) = \frac{C_\epsilon}{C} \tag{6.7}$$

where $C_k$ and $C_\epsilon$ denote the areas of the regions $B_k \cap B$ and $B_\epsilon \cap B$, respectively.

As a consequence:

$$\begin{aligned} \mathbb{E}[N\mathbf{1}_{A_k}] &= \frac{1}{C}\int\int_{B_k\cap B} \frac{\lambda}{\mu-\lambda} d\lambda d\mu \\ \mathbb{E}[N|A_k] &= \frac{1}{C_k}\int\int_{B_k\cap B} \frac{\lambda}{\mu-\lambda} d\lambda d\mu \end{aligned} \tag{6.8}$$

and

$$\begin{aligned} \mathbb{E}[N^2\mathbf{1}_{A_k}] &= \frac{1}{C}\int\int_{B_k\cap B} \frac{\lambda^2}{(\mu-\lambda)^2} d\lambda d\mu \\ \mathbb{E}[N^2|A_k] &= \frac{1}{C_k}\int\int_{B_k\cap B} \frac{\lambda^2}{(\mu-\lambda)^2} d\lambda d\mu \end{aligned} \tag{6.9}$$

Eqs. (6.8) and (6.9) can be easily computed, but the results depend on the relative positions of the endpoints of $[\lambda_L, \lambda_U]$ and $[\mu_L, \mu_U]$ intervals. An explicit expression can be obtained in all cases, but here only one is shown, for which the integrals are stepwise evaluated.

Clearly, all these formulas result in closed-form expressions because of the choice of uniform densities for the random variables. Numerical integration could become necessary if different and more complex choices were made.

Explicit computations are shown in the following for $\lambda_L < \mu_L < \mu_U < \lambda_U$. In this case, the area of the integration region is

$$C_k = \frac{1}{2}(\mu_U - \mu_L)\left(\frac{\mu_U}{k} - \lambda_L + \frac{\mu_L}{k} - \lambda_L\right),$$

and the integral in Eq. (6.8) becomes:

$$\begin{aligned}
&\int\int_{B_k\cap B}\frac{\lambda}{\mu-\lambda}d\lambda d\mu \\
&\qquad=\int\int_{B_k\cap B}\left(\frac{\mu}{\mu-\lambda}-1\right)d\lambda d\mu \\
&\qquad=\int_{\mu_L}^{\mu_U}\int_{\lambda_L}^{\frac{\mu}{k}}\frac{\mu}{\mu-\lambda}d\lambda d\mu - C_k \\
&\qquad=-\int_{\mu_L}^{\mu_U}\mu\ln(\mu-\lambda)\Big|_{\lambda_L}^{\frac{\mu}{k}}d\mu - C_k \\
&\qquad=-\int_{\mu_L}^{\mu_U}\mu\ln\left(\frac{k-1}{k}\mu\right)d\mu+\int_{\mu_L}^{\mu_U}\mu\ln(\mu-\lambda_L)d\mu - C_k
\end{aligned} \tag{6.10}$$

Eq. (6.10) is easily computable in an explicit manner, leading to:

$$\begin{aligned}
&\int\int_{B_k\cap B}\frac{\lambda}{\mu-\lambda}d\lambda d\mu \\
&\qquad=\frac{\mu_U^2}{2}\ln\left(\frac{k(\mu_U-\lambda_L)}{(k-1)\mu_U}\right)-\frac{\mu_L^2}{2}\ln\left(\frac{k(\mu_L-\lambda_L)}{(k-1)\mu_L}\right) \\
&\qquad\quad-\frac{\lambda_L}{2}(\mu_u-\mu_L)-\frac{\lambda_L^2}{2}\ln\left(\frac{\mu_U-\lambda_L}{\mu_L-\lambda_L}\right)-C_k
\end{aligned} \tag{6.11}$$

The second moment of the random variable $N$ is similarly derived:

$$\begin{aligned}
&\int\int_{B_k\cap B}\frac{\lambda^2}{(\mu-\lambda)^2}d\lambda d\mu \\
&\qquad=C_k+\mu_U^2\ln\left(\frac{(k-1)\mu_U}{k(\mu_U-\lambda_L)}\right) \\
&\qquad\quad-\mu_L^2\ln\left(\frac{(k-1)\mu_L}{k(\mu_L-\lambda_L)}\right)-\frac{k-2}{k-1}(\mu_U^2-\mu_L^2)
\end{aligned} \tag{6.12}$$

Assembling all pieces together an explicit expression for the expectation and the variance of the steady state average number of entities in the system is obtained.

Computations run along the same lines also to compute:

$$\begin{aligned}
&\int_{\mathbb{R}}\int_{\mathbb{R}} \frac{\lambda}{\mu-\lambda}\mathbf{1}_{\{\mu>\lambda+\epsilon\}} f_{\Lambda,M}(\lambda,\mu)d\lambda d\mu \\
&\quad= \int_{\lambda_L}^{\lambda_U}\int_{\lambda+\epsilon}^{\mu_U} \frac{\lambda}{\mu-\lambda} d\mu d\lambda \\
&\quad= \frac{(\mu_U^2-\lambda_L^2)}{2}\ln(\mu_U-\lambda_L) - \frac{(\mu_U^2-\lambda_U^2)}{2}\ln(\mu_U-\lambda_U) \\
&\qquad - (\lambda_U-\lambda_L)\left[\frac{1}{4}(\lambda_U+\lambda_L)+\frac{\mu_U}{2}+\ln\epsilon\right]
\end{aligned} \tag{6.13}$$

All the other cases may be similarly computed and are described in Section 6.3.1.

In the remainder of this section the behavior of the model proposed is studied while the amount of uncertainty decreases and the integration areas defined above collapse towards a single point.

We assume the Uniform density to be concentrated on the rectangle $[\lambda_L,\lambda_U]\times[\mu_L,\mu_U]$ as the dimensions $\delta_1=\lambda_U-\lambda_L$, $\delta_2=\mu_U-\mu_L$ tend to 0, i.e., $\lambda_L,\lambda_U\to\lambda_0$ and $\mu_L,\mu_U\to\mu_0$, where by $(\lambda_0,\mu_0)$ we are denoting any point such that $\mu_0>\lambda_0$.

In this case, for $\delta_1$ and $\delta_2$ sufficiently small, the integration region is completely contained into the limitation region $A_k$ or $A_\epsilon$, hence the conditional expectation is given by:

$$\mathbb{E}[N|A_k] = \frac{1}{\delta_1\delta_2}\int_{\lambda_L}^{\lambda_U}\int_{\mu_L}^{\mu_U}\frac{\lambda}{\mu-\lambda}d\mu d\lambda \tag{6.14}$$

and we want to estimate the error made by substituting this expression with the corresponding certainty expression $\lambda_0/(\mu_0-\lambda_0)$.

Applying the changes of variable $\lambda=x+\lambda_L$ and $\mu=y+\mu_L$, Eq. (6.14) becomes:

$$G(\delta_1,\delta_2) = \int_0^{\delta_1}\int_0^{\delta_2}\frac{x+\lambda_L}{y-x+\mu_L-\lambda_L}dydx \tag{6.15}$$

Eq. (6.15) is a smooth function in two variables that can be expanded

around $(0,0)$ up to the third order obtaining:

$$\begin{aligned} G(\delta_1,\delta_2) = \\ &= G(0,0) + \frac{\partial G}{\partial \delta_1}(0,0)\delta_1 + \frac{\partial G}{\partial \delta_2}(0,0)\delta_2 \\ &+ \frac{1}{2}\left[\frac{\partial^2 G}{\partial \delta_1^2}(0,0)\delta_1^2 + \frac{\partial^2 G}{\partial \delta_2^2}(0,0)\delta_2^2 + 2\frac{\partial^2 G}{\partial \delta_1 \partial \delta_2}(0,0)\delta_1\delta_2\right] \\ &+ \frac{1}{3!}\left[\frac{\partial^3 G}{\partial \delta_1^3}(0,0)\delta_1^3 + 3\frac{\partial^3 G}{\partial \delta_1^2 \partial \delta_2}(0,0)\delta_1^2\delta_2\right. \\ &\quad \left. +3\frac{\partial^3 G}{\partial \delta_1 \partial \delta_2^2}(0,0)\delta_1\delta_2^2 + \frac{\partial^3 G}{\partial \delta_2^3}(0,0)\delta_2^3\right] \\ &+ o(p_4(\delta_1,\delta_2)) \end{aligned} \tag{6.16}$$

where $p_4(\delta_1,\delta_2)$ indicates a polynomial of degree 4 in $\delta_1,\delta_2$.

It is straightforward to see that $G(0,0)=0$ and for any $n \geq 1$:

$$\begin{aligned} \frac{\partial^n G}{\partial \delta_1^n}(\delta_1,\delta_2) &= \int_0^{\delta_2} \frac{\partial^{n-1}}{\partial \delta_1^{n-1}}\left(\frac{\delta_1+\lambda_L}{y-\delta_1+\mu_L-\lambda_L}\right) dy \;\Rightarrow\; \frac{\partial^n G}{\partial \delta_1^n}(0,0) = 0 \\ \frac{\partial^n G}{\partial \delta_2^n}(\delta_1,\delta_2) &= \int_0^{\delta_1} \frac{\partial^{n-1}}{\partial \delta_2^{n-1}}\left(\frac{x+\lambda_L}{\delta_2-x+\mu_L-\lambda_L}\right) dx \Rightarrow \frac{\partial^n G}{\partial \delta_2^n}(0,0) = 0 \end{aligned} \tag{6.17}$$

hence, substituting Eq. (6.17) into Eq. (6.16):

$$\begin{aligned} G(\delta_1,\delta_2) = \\ &= \frac{\partial^2 G}{\partial \delta_1 \partial \delta_2}(0,0)\delta_1\delta_2 \\ &\quad + \frac{1}{2}\left[\frac{\partial^3 G}{\partial \delta_1^2 \partial \delta_2}(0,0)\delta_1^2\delta_2 + \frac{\partial^3 G}{\partial \delta_1 \partial \delta_2^2}(0,0)\delta_1\delta_2^2\right] \\ &\quad + o(\delta_1\delta_2) \end{aligned} \tag{6.18}$$

Computing the three terms we obtain:

$$\begin{aligned} \frac{\partial^2 G}{\partial \delta_1 \partial \delta_2}(0,0) &= \frac{\lambda_L}{\mu_L-\lambda_L} \\ \frac{\partial^3 G}{\partial \delta_1^2 \partial \delta_2}(0,0) &= \frac{\mu_L}{(\mu_L-\lambda_L)^2} \\ \frac{\partial^3 G}{\partial \delta_1 \partial \delta_2^2}(0,0) &= \frac{-\lambda_L}{(\mu_L-\lambda_L)^2} \end{aligned} \tag{6.19}$$

and plugging Eq. (6.19) into Eq. (6.18):

$$\frac{G(\delta_1,\delta_2)}{\delta_1\delta_2} = \frac{\lambda_L}{\mu_L-\lambda_L} + \frac{1}{(\mu_L-\lambda_L)^2}(\mu_L\delta_1 - \lambda_L\delta_2) + o(\delta_1+\delta_2) \quad (6.20)$$

As a matter of fact, the function $G$ depends also on the two parameters $\lambda_L$ and $\mu_L$ which respectively tend to $\lambda_0$ and $\mu_0$ as $\delta_1 \to 0$ and $\delta_2 \to 0$. The error made by evaluating the expression directly at $\lambda_0$ and $\mu_0$ is:

$$\frac{|G(\delta_1,\delta_2,\lambda_L,\mu_L) - G(\delta_1,\delta_2,\lambda_0,\mu_0)|}{\delta_1\delta_2}.$$

Assuming that $\lambda_0-\lambda_L = \kappa\delta_1$ and $\mu_0-\mu_L = h\delta_2$ for some $0 \le \kappa, h \le 1$ and applying the mean value theorem we get

$$\frac{|G(\delta_1,\delta_2,\lambda_L,\mu_L) - G(\delta_1,\delta_2,\lambda_0,\mu_0)|}{\delta_1\delta_2}$$
$$\le$$
$$\left|\frac{\partial G(\delta_1,\delta_2,\xi,\mu_0)}{\partial\lambda_L}\right|\kappa\delta_1 + \left|\frac{\partial G(\delta_1,\delta_2,\lambda_0,\zeta)}{\partial\mu_L}\right|h\delta_2$$

for appropriately chosen $\xi \in [\lambda_L,\lambda_0]$ and $\zeta \in [\mu_L,\mu_0]$. After some calculations it can be shown that, in the given initial intervals, both partial derivatives can be uniformly bounded by

$$\mu_U\left[\frac{1}{(\mu_L-\lambda_U)^2} + \frac{\delta_1+\delta_2}{(\mu_L-\lambda_U)^3} + \frac{2\delta_1^2+2\delta_2^2+3\delta_1\delta_2}{(\mu_L-\lambda_U)^4}\right].$$

Summarizing,

$$\frac{|G(\delta_1,\delta_2,\lambda_L,\mu_L)|}{\delta_1\delta_2} = \frac{\lambda_0}{\mu_0-\lambda_0} + o(\delta_1+\delta_2)$$

as expected.

### 6.2.2 The Erlang case

When dealing with M/M/1 queues (i.e., exponential distributed arrival and service times), it is possible to show that the arrival and service rates, $\Lambda$ and $M$, are Erlang distributed with as many stages as the number of collected samples for inter-arrival and service times, respectively [133]. For instance, considering the inter-arrival time $X$, the sum $S$ of a random sample of size $n$ (i.e., $S = \sum_{i=1}^{r} X_i$) is $r$-stage Erlang distributed with rate parameter $\lambda$, and hence the probability density function of $S$ is [133]:

$$f_{S|\Lambda}(s|\lambda) = \frac{\lambda^r s^{r-1} e^{-\lambda s}}{(r-1)!} \quad (6.21)$$

As shown in [99], applying Bayes' theorem and using Jeffreys' prior for $\Lambda$ (i.e., $f_\Lambda(\lambda) = s/\lambda$), the pdf of the rate $\Lambda$ is $r$-stage Erlang with rate $s$ and has the form:

$$f_\Lambda(\lambda) = f_{\Lambda|S}(\lambda|s) = \frac{\lambda^{r-1} s^r e^{-\lambda s}}{(r-1)!}. \tag{6.22}$$

In the same way, for the service time $Y$ with rate parameter $\mu$, it can be shown that $U$ follows a $n$-stage Erlang distribution with rate $\mu$ such that $U = \sum_{i=1}^{n} Y_i$. Hence, rate $M$ as a random variable is $n$-stage Erlang distributed with rate parameter $u$.

Thus, the joint probability density of the two independent random variables is:

$$f_{\Lambda,M}(\lambda,\mu) = f_\Lambda(\lambda) f_M(\mu) = \frac{\lambda^{r-1} s^r e^{-\lambda s}}{(r-1)!} \cdot \frac{\mu^{n-1} u^n e^{-\mu u}}{(n-1)!} \tag{6.23}$$

where $\Lambda$ follows an $Erlang(r, s)$ density, with $r \in \mathbb{N}$, $s > 0$, while $M$ is another $Erlang(n, u)$ distributed random variable, $n \in \mathbb{N}$, $u > 0$. The area of integration is no more limited by the bounds of the input parameters, and their samples are in the interval $[0, \infty)$, but need to satisfy the stability constraint (i.e., $\lambda < \mu$). In this setting, the probability of $k$ limitation becomes:

$$\begin{aligned} P(A_k) &= \int_0^\infty \int_0^{\frac{\mu}{k}} \frac{\lambda^{r-1} s^r e^{-\lambda s}}{(r-1)!} \cdot \frac{\mu^{n-1} u^n e^{-\mu u}}{(n-1)!} d\lambda d\mu \\ P(A_k) &= 1 - \Big(\frac{uk}{s+uk}\Big)^n \sum_{i=0}^{r-1} \binom{n+i-1}{n-1} \Big(\frac{s}{s+ku}\Big)^i \end{aligned} \tag{6.24}$$

Similarly for region $A_\epsilon$ we have:

$$\begin{aligned} P(A_\epsilon) &= \int_\epsilon^\infty \int_0^{\mu-\epsilon} \frac{\lambda^{r-1} s^r e^{-\lambda s}}{(r-1)!} \cdot \frac{\mu^{n-1} u^n e^{-\mu u}}{(n-1)!} d\lambda d\mu \\ &= 1 - e^{\epsilon s} \Big(\frac{u}{s+u}\Big)^n \sum_{i=0}^{r-1} \sum_{j=0}^{i} \binom{n+j-1}{n-1} \frac{(-\epsilon s)^{i-j}}{(i-j)!} \Big(\frac{s}{s+u}\Big)^j \end{aligned} \tag{6.25}$$

The cumulative distribution function, expected values and second moments,

when $k$ is used to limit the values that $\Lambda$ can assume, are:

$$\begin{aligned} F_{N|A_k}(x) &= \iint_{B_k} \mathbf{1}_{\{\frac{\lambda}{\mu-\lambda} \leq x\}} f_\Lambda(\lambda) f_M(\mu) d\lambda d\mu, \qquad x \in \mathbb{R} \\ \mathbb{E}[N|A_k] &= \frac{1}{P(A_k)} \cdot \iint_{B_k} \frac{\lambda}{\mu-\lambda} f_\Lambda(\lambda) f_M(\mu) d\lambda d\mu \\ \mathbb{E}[N^2|A_k] &= \frac{1}{P(A_k)} \cdot \iint_{B_k} \frac{\lambda^2}{(\mu-\lambda)^2} f_\Lambda(\lambda) f_M(\mu) d\lambda d\mu \end{aligned} \tag{6.26}$$

and

$$\begin{aligned} F_{R|A_k}(x) &= \iint_{B_k} \mathbf{1}_{\{\frac{1}{\mu-\lambda} \leq x\}} f_\Lambda(\lambda) f_M(\mu) d\lambda d\mu, \qquad x \in \mathbb{R} \\ \mathbb{E}[R|A_k] &= \frac{1}{P(A_k)} \cdot \iint_{B_k} \frac{1}{\mu-\lambda} f_\Lambda(\lambda) f_M(\mu) d\lambda d\mu \\ \mathbb{E}[R^2|A_k] &= \frac{1}{P(A_k)} \cdot \iint_{B_k} \frac{1}{(\mu-\lambda)^2} f_\Lambda(\lambda) f_M(\mu) d\lambda d\mu \end{aligned} \tag{6.27}$$

for $N$ and $R$, respectively. Similar equations can be obtained using the $\epsilon$-limitation strategy.

Differently from the Uniform case, these expressions are not easily integrable and numerical integration is used to solve them. Note that, in the case of Erlang distribution, Eqs. (6.26) and (6.27) can be correctly used for all the values of $\lambda$ and $\mu$, while the Uniform case requires a different expression depending on the relative values of $\lambda_L$, $\lambda_U$, $\mu_L$ and $\mu_U$.

## 6.3 Experiments and comparison of the approaches

This section provides further details of the derivations based on the basic framework given in Section 6.2 for different Uniform (Section 6.3.1) and Erlang (Section 6.3.2) alternatives. The two proposed alternatives are compared in Section 6.3.3.

### 6.3.1 Uniform density results

Considerations about the derivations of $\mathbb{E}[N|A_{k,\epsilon}]$, $\mathbb{E}[R|A_{k,\epsilon}]$, $Var[N|A_{k,\epsilon}]$ and $Var[R|A_{k,\epsilon}]$ are reported in this section in case of Uniform distribution, while considering different relative positions of $\lambda$ and $\mu$'s lower and upper bounds (i.e., $\lambda_L$, $\lambda_U$, $\mu_L$, $\mu_U$). From now on, to simplify notation, we indicate those as $\mathbb{E}[N]$, $\mathbb{E}[R]$, $Var[N]$, and $Var[R]$. In order to cope with all possibilities, five different cases have to be considered:

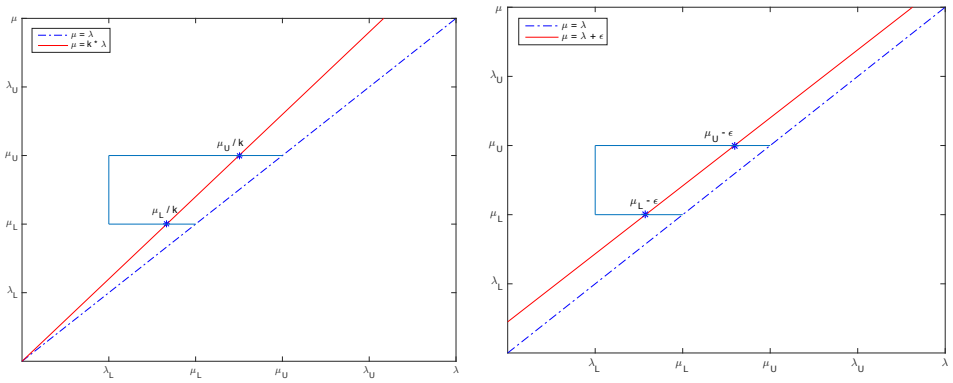



**Figure 6.1:** *Case (3):* $\lambda_L < \mu_L < \mu_U < \lambda_U$*, for (a)* $\mu = k \cdot \lambda$ *and (b)* $\mu = \lambda + \epsilon$.

(1) $\lambda_L < \lambda_U < \mu_L < \mu_U$;

(2) $\mu_L < \lambda_L < \mu_U < \lambda_U$;

(3) $\lambda_L < \mu_L < \mu_U < \lambda_U$;

(4) $\mu_L < \lambda_L < \lambda_U < \mu_U$;

(5) $\lambda_L < \mu_L < \lambda_U < \mu_U$.

For the sake of order, graphical representation is reported only for case (3). In this case the straight line intersects the integration area, thus further elaborations are needed to evaluate the integral. Specifically, this case is analyzed by considering the limitations defined in Section 6.2, i.e.: *i)* $\mu > k\lambda$, that is referred to as *prod*, and *ii)* $\mu > \lambda + \epsilon$, that is referred to as *sum*. Figure 6.1a reports the *prod* case, where the intersection with the line $\mu = k\lambda$ occurs in points $(\mu_L/k, \mu_L)$ and $(\mu_U/k, \mu_U)$. Instead, Figure 6.1b shows the *sum* case, where the intersection with the line $\mu = \lambda + \epsilon$ occurs in points $(\mu_L - \epsilon, \mu_L)$ and $(\mu_U - \epsilon, \mu_U)$. These points are then used in the formulas for $\mathbb{E}[N]$, $\mathbb{E}[R]$, $Var[N]$, and $Var[R]$ since they delimit the area of interest.

Closed-form formulas for deriving $\mathbb{E}[N]$, $\mathbb{E}[R]$, $Var[N]$, and $Var[R]$ of all cases have been derived using Mathematica [74]. $\mathbb{E}[N]$ for case (3) with $k$ and $\epsilon$ limitations is given in Eqs. (6.12) and (6.13), respectively; for the sake of order, the formulas of all other cases are omitted.

However, although some cases have more complex formulas than others, it is worth noting that while considering $\Lambda$ and $M$ as uniformly distributed in the ranges $(\lambda_L,\lambda_U)$ and $(\mu_L,\mu_U)$, it is possible to calculate the average

number of entities and the average response time along with their variances by means of closed-form formulas in all cases, as claimed in Section 6.2.

### 6.3.2 Erlang density results

The results obtained with Erlang distributed rate parameters, as described in Section 6.2.2, are shown. Also in this case, as for the Uniform one, both the additive and the multiplicative limitation techniques are considered. To allow a fair comparison between the Uniform and the Erlang alternatives, we select some combinations of the Erlang distributed rate parameters to obtain $5$ cases similar to the ones identified in Section 6.3.1. In particular, parameters have been chosen in a way such that values $\lambda_L$ and $\lambda_U$ can be interpreted as a confidence interval on $\Lambda$, and similarly for $M$. Being $\hat{\lambda}$ the point estimate for random variable $\Lambda$, such that $\hat{\lambda} = \sum_{i=1}^{r} \lambda_i / r$, where $\lambda_i$ are the values of the same random variable. The following assumptions have been used:

1. $\mathbb{E}\left[\Lambda \sim Erlang\left(r, \frac{r}{\hat{\lambda}}\right)\right] = \frac{\lambda_L + \lambda_U}{2}$
2. $r = min\left\{h \in \mathbb{N} : P\left(\lambda_L \leq \Lambda \sim Erlang\left(h, \frac{h}{\hat{\lambda}}\right) \leq \lambda_U\right) \geq 1 - \alpha\right\}$

The first assumption specifies that the expected values of the inter-arrival rate in the Erlang case must be equal to the expected value of the same parameter with uniform distribution. The second one sets the confidence interval of rate parameter between $\lambda_L$ and $\lambda_U$ (the parameters of the Uniform distribution) with a $100(1-\alpha)\%$ confidence level, by choosing $r$ as the minimum integer that concentrates at least $1-\alpha$ probability mass of an Erlang distribution, with the given mean between $\lambda_L$ and $\lambda_U$. Similar assumptions can be done for the service rate $M$. Following [98], the value of $r$ can be determined analytically:

$$r = \left\lceil \frac{\hat{\lambda}}{4 d_\lambda} \{\chi^2_{2r,\alpha/2} - \chi^2_{2r,1-\alpha/2}\} \right\rceil \tag{6.28}$$

where $\chi^2_{2r,1-\alpha/2}$ is the critical value of the chi-square distribution with $2r$ degrees of freedom, $\hat{\lambda} = \frac{\lambda_U + \lambda_L}{2}$, and $d_\lambda = \frac{\lambda_U - \lambda_L}{2}$ is the half-width of the confidence interval.

The analysis is performed for different confidence interval lengths of each input parameter; for example, for a given inter-arrival rate $\hat{\lambda}$, and its half-width confidence interval $d_\lambda$, we set $\lambda_L = \hat{\lambda} - d_\lambda$ and $\lambda_U = \hat{\lambda} + d_\lambda$ for the Uniform case, and the number of stages of the Erlang case is computed

using Eq. (6.28). Similar considerations are done for the service rate $M$. In order to comply with the considered case, also the values of the rate parameters may change. In particular, for case (1), case (3), case (4) and case (5) the average rates are $\hat{\lambda} = 0.4$ and $\hat{\mu} = 0.6$, whereas in case (2) $\hat{\lambda} = 0.6$ and $\hat{\mu} = 0.55$. Note that, even if in this last case the system is unstable, it is still possible to get non-divergent results when the confidence intervals of the two rate parameters are large enough to include cases in which $\lambda < \mu$. Also in this section, $\mathbb{E}[N]$ and $Var[N]$ are depicted only for case (3) (i.e., $\lambda_L < \mu_L < \mu_U < \lambda_U$). These results with both $\epsilon$ and $k$ limitations are plotted in Figure 6.2.

For all output metrics and for all cases, $\epsilon$-limitation provides results that are closer to the one of a system where the parameters are not subject to uncertainty (see Section 6.4 for further considerations about that). The only exception occurs in case (1) and for tight confidence intervals, where the two limitations seem to provide similar results. For all cases except case (2), the tighter the confidence intervals of the two rate parameters (i.e., the larger is the number of samples that are collected for the two parameters), the more accurate the output metrics, and their expected values are closer to the exact ones. This property is further stressed by the fact that the corresponding variance tends to $0$.

Since case (2) is the only one dealing with an unstable system, the variance of its output metrics is closer to $0$ when the confidence intervals of the rates is larger. That is because when measures are less accurate, service rate may be larger than the inter-arrival one, and the system is still stable. Note also that, although expected values and variances of random variables $N$ and $R$ are different, their behaviors are similar.

Finally, based on the considered cases, one of the two rate parameters may affect the output metrics more than the other one. This is due to the confidence interval length of each input parameter. For example, in case (1), case (3) and case (5), the confidence interval length of the inter-arrival rate is varying faster than the confidence interval of the service rate, and both $N$ and $R$ seem to depend more on inter-arrival rate. The opposite happens in case (4), where the two output random variables are more affected by the service rate. Instead, in case (2), the confidence intervals length is the same for both inter-arrival and service rates.

### 6.3.3 Comparison between Uniform and Erlang results

After analyzing both the proposed alternatives (i.e., Uniform and Erlang), we now compare them to identify the gap among the provided results. As

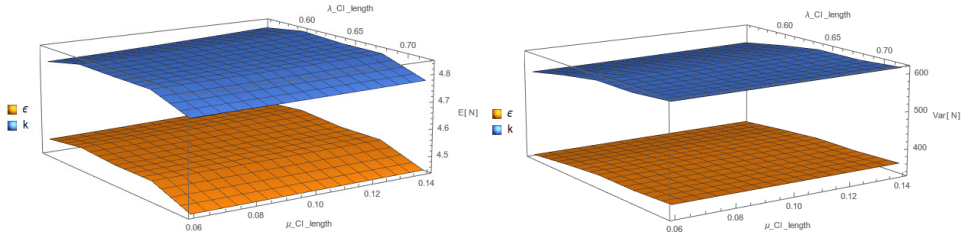



**Figure 6.2:** *(a) Expectation and (b) variance of random variable $N$ for the Erlang distributed rates while considering case (3), with $\epsilon$ and $k$ limitation techniques.*

previously said, the Uniform and the Erlang alternatives are meant for different scenarios: the Uniform one may be used when no information about the rate parameters are given, whereas Erlang densities are adopted if inter-arrival and service times are exponentially distributed.

The comparison between the two alternatives is performed assuming that the average inter-arrival and service rates are $\hat{\lambda} = 0.4\ j/s$ and $\hat{\mu} = 0.6\ j/s$, respectively, and varying the length of their confidence interval. In particular, the confidence interval half-width of inter-arrival rate is $d_\lambda = [1, 10] \cdot 4 \cdot 10^{-2}$, and $d_\mu = [1, 10] \cdot 10^{-2}$ is the one of service rate. The model is analyzed under some of the cases previously identified (i.e., case (1), case (3) and case (5)) while changing the $d_\lambda$ and $d_\mu$ values. Cases (2) and (4) can be similarly generated and analyzed.

Table 6.1 reports an excerpt of the calculation performed for $\mathbb{E}[N]$ while using the closed formulas introduced in Section 6.3.1. The table is structured as follows: the rows report the actual values of $\lambda_L$ and $\lambda_U$, whereas on the columns the ones used for $\mu_L$ and $\mu_U$. These values are derived by varying the $d_\lambda$ and $d_\mu$ values in the ranges defined above. Entries of the table indicate the $\mathbb{E}[N]$ values obtained while adopting the appropriate formula corresponding to one of the cases presented in Section 6.3.1. Specifically, the cells of Table 6.1 have different colors indicating the used formula: light gray for case (1), white for case (5), and dark gray for case (3). Table 6.1a reports the $\mathbb{E}[N]$ values while considering the $k$-limitation, whereas Table 6.1b shows the $\mathbb{E}[N]$ values for the $\epsilon$-one.

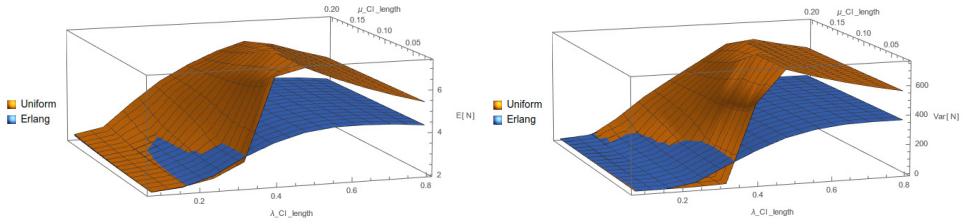
**Table 6.1:** *Closed formulas applied for the evaluation of $\mathbb{E}[N]$ with (a) multiplicative and (b) additive limitations. Three cases are analyzed: case (1) is light gray, case (3) is gray and case (5) is white.*

**(a)**

| | | | $(\mu_L, \mu_U)$ | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | d=0.05 | d=0.06 | d=0.07 | d=0.08 | d=0.09 | d=0.10 |
| | | | (0.55, 0.65) | (0.54, 0.66) | (0.53, 0.67) | (0.52, 0.68) | (0.51, 0.69) | (0.5, 0.7) |
| $(\lambda_L, \lambda_U)$ | d=0.01 | (0.36, 0.44) | 2.08919 | 2.11184 | 2.13986 | 2.17409 | 2.21562 | 2.26604 |
| | d=0.02 | (0.32, 0.48) | 2.24561 | 2.27889 | 2.32137 | 2.37550 | 2.44534 | 2.53845 |
| | d=0.03 | (0.28, 0.52) | 2.60534 | 2.68476 | 2.80566 | 3.06077 | 3.48777 | 3.82179 |
| | d=0.04 | (0.24, 0.56) | 4.12062 | 4.43937 | 4.65711 | 4.81292 | 4.92817 | 5.01549 |
| | d=0.05 | (0.2, 0.6) | 6.32117 | 6.20215 | 6.10217 | 6.01597 | 5.94016 | 5.87244 |
| | d=0.06 | (0.16, 0.64) | 7.75326 | 7.43185 | 7.15425 | 6.91942 | 6.71934 | 6.54677 |
| | d=0.07 | (0.12, 0.68) | 7.36737 | 7.36733 | 7.36728 | 7.36723 | 7.23214 | 7.03606 |
| | d=0.08 | (0.08, 0.72) | 6.81550 | 6.81549 | 6.81547 | 6.81545 | 6.81543 | 6.81540 |
| | d=0.09 | (0.04, 0.76) | 6.33616 | 6.33616 | 6.33615 | 6.33615 | 6.33615 | 6.33614 |
| | d=0.10 | (0.0, 0.8) | 5.91566 | 5.91566 | 5.91566 | 5.91566 | 5.91566 | 5.91566 |

**(b)**

| | | | $(\mu_L, \mu_U)$ | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | d=0.05 | d=0.06 | d=0.07 | d=0.08 | d=0.09 | d=0.10 |
| | | | (0.55, 0.65) | (0.54, 0.66) | (0.53, 0.67) | (0.52, 0.68) | (0.51, 0.69) | (0.5, 0.7) |
| $(\lambda_L, \lambda_U)$ | d=0.01 | (0.36, 0.44) | 2.08919 | 2.11184 | 2.13986 | 2.17409 | 2.21562 | 2.26604 |
| | d=0.02 | (0.32, 0.48) | 2.24561 | 2.27889 | 2.32137 | 2.37550 | 2.44534 | 2.53845 |
| | d=0.03 | (0.28, 0.52) | 2.60534 | 2.68476 | 2.80566 | 3.05430 | 3.40319 | 3.67364 |
| | d=0.04 | (0.24, 0.56) | 4.01118 | 4.26172 | 4.42909 | 4.54557 | 4.62878 | 4.68914 |
| | d=0.05 | (0.2, 0.6) | 5.90743 | 5.78397 | 5.67928 | 5.58821 | 5.50743 | 5.43467 |
| | d=0.06 | (0.16, 0.64) | 7.12138 | 6.84135 | 6.59190 | 6.37700 | 6.19142 | 6.02958 |
| | d=0.07 | (0.12, 0.68) | 6.73470 | 6.73530 | 6.73601 | 6.73683 | 6.62967 | 6.45721 |
| | d=0.08 | (0.08, 0.72) | 6.23117 | 6.23174 | 6.23242 | 6.23321 | 6.23410 | 6.23510 |
| | d=0.09 | (0.04, 0.76) | 5.79330 | 5.79384 | 5.79449 | 5.79523 | 5.79607 | 5.79702 |
| | d=0.10 | (0.0, 0.8) | 5.40877 | 5.40928 | 5.40988 | 5.41058 | 5.41137 | 5.41226 |




(a) (b)

**Figure 6.3:** *Comparison of Uniform and Erlang (a) expectations and (b) variances of random variable $N$, adopting $\epsilon$-limitation.*

The Uniform and the Erlang alternatives have been compared by studying the expected values and the variances calculated for the number of entities and the system response time (i.e., $\mathbb{E}[N]$, $\mathbb{E}[R]$, $Var[N]$ and $Var[R]$). For the considered set of parameters, the exact values of the two considered measures are $N = 2$ entities and $R = 5$ seconds. Furthermore, both $\epsilon$ and $k$ limitations are taken into account, assuming $\epsilon = 0.001$ and $k = 1.001$. For the sake of order, only the results for $N$ when using $\epsilon$-limitation are depicted in Figure 6.3, where the Uniform and Elrnag alternatives are compared. Note that, similar results were obtained using $k$-limitation.

If the confidence intervals of the two rate parameters are tight enough, the Uniform alternative performs better than the Erlang one. Indeed, the expected values of the output metric computed with the two different alternatives are close, while the variance obtained with the Erlang alternative is slightly larger than the one showed by the Uniform one. It is important to note that, since the variance is related to the confidence interval, the smaller the variance, the tighter the confidence interval of the analyzed output metric. The Erlang alternative performs better than the Uniform one for input parameters with less tight confidence interval. Both the expected values and the variances computed with Erlang alternative are smaller than those computed with the Uniform one. Similar results are obtained while considering random variable $R$.

A further analysis is performed considering the length of the two-sided $95\%$ confidence intervals for both the output metrics. In particular, the cumulative distribution functions of $N$ and $R$ are computed both with Uniform and Erlang alternatives. Then, to derive the confidence interval of the two output random variables, the $2.5th$ and $97.5th$ percentiles for each Cdf are derived.

The results are shown in Figure 6.4, that depicts the two-sided $95\%$

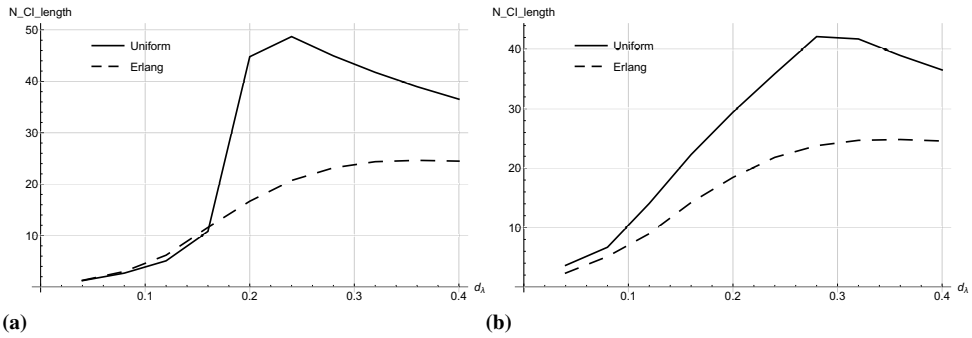



**Figure 6.4:** *Confidence interval length of output random variable $N$ when (a) $d_\mu = 0.01$ and (b) $d_\mu = 0.1$.*

confidence interval lengths of random variable $N$, when derived with $\epsilon$-limitation, against confidence interval length of inter-arrival rate. In particular, they are obtained assuming $\hat{\lambda} = 0.4\ jobs/s$ and $\hat{\mu} = 0.6\ jobs/s$, and fixing the half-width of confidence interval of service rate to $0.01$ (Figure 6.4a) and to $0.1$ (Figure 6.4b). The results show that the Erlang alternative provides the tighter confidence interval for output metric $N$, especially when confidence interval on inter-arrival rate is larger. As previously said, the Uniform alternative can work slightly better than the Erlang one if confidence intervals of both inter-arrival and service rates are tight enough. Similar results have been obtained for system response time, $R$.

Similar results may be obtained also for different values of arrival and service rates. For example, the results for number of customers, $N$, when $\hat{\lambda} = 0.6\ jobs/s$ and $\hat{\mu} = 1\ jobs/s$, are shown in Figure 6.5. In particular, Figure 6.5a shows the $95\%$ confidence interval for computed with the two different techniques. They are depicted as a function of half-width confidence interval on input parameters (i.e., $d_\lambda$ and $d_\mu$) that, for the sake of simplicity, are assumed to be the same. In this case, the Erlang technique is providing more accurate results than the Uniform one, indeed the confidence interval obtained with the Erlang approach is tighter than that from the Uniform one. Instead, Figure 6.5b validates the approach presented in this chapter, generating the model's input parameters from exponential distribution. For this reason, only the Erlang technique has been considered. In fact, each point in the figure is obtained considering a different amount of observations to estimate the input parameters of the model. In this case, each curve is depicted as a function of the number of collected samples for each input parameter (i.e., $k_\lambda$ and $k_\mu$). As visible, each point is within the

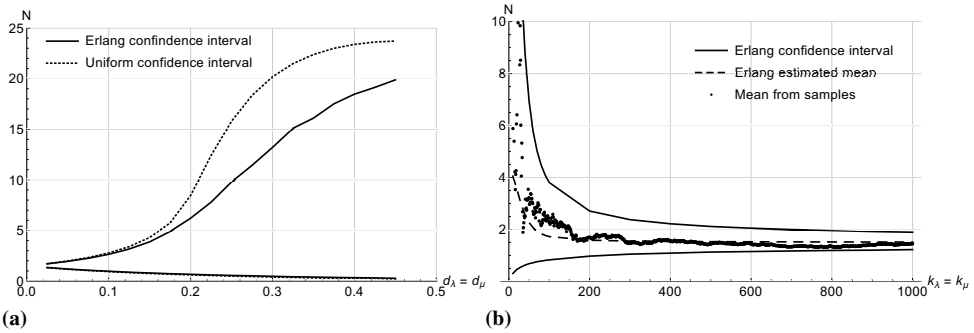



**Figure 6.5:** *Average number of customers, $N$, in the M/M/1 queue, , when $\hat{\lambda} = 0.6$ j/s and $\hat{\mu} = 1$ j/s, for (a) comparing the Uniform and Erlang approaches and (b) validating the Erlang one.*

confidence interval bounds, proving that the technique is providing good results.

To fairly compare the two alternatives, the time they require to estimates expected values and variances of the output measures must be also taken into account. In fact, closed-form equations have been provided for the Uniform alternative, thus the time to get results with this strategy is negligible. Instead, as said in Section 6.2.2, the Erlang equations are not easly treatable and numerical integration (i.e., numerical-solution) has been used to study $N$ and $R$. In this case, the time needed by the Erlang alternative to provide results is not negligible, since every time input parameters change, the result must be computed from the beginning. For example, Figure 6.6 shows the time the Erlang alternative requires to compute $Var[N]$ (i.e., first and second moments) with $\epsilon$-limitation. The tighter the confidence intervals on the input parameters, the longer the time needed for computation. Similar results have been obtained with $k$-limitation and they are not reported here for the sake of simplification. In particular, the total time required by Erlang alternative for the computation of all the (Erlang) values shown in Figure 6.3b has been slightly longer than $65$ seconds. Similar considerations may be drawn for the computation of $Var[R]$.

Summarizing, the Uniform alternative is always faster than the Erlang one, since closed-form expressions have been derived for this alternative. Nonetheless, if the confidence intervals of the input parameters are not tight enough, it could be the case to sacrifice the performance for more accurate results.

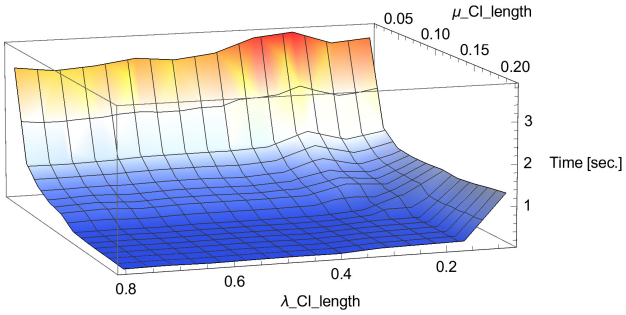



**Figure 6.6:** *Time required by the Erlang alternative to compute $Var[N]$ with $\epsilon$-limitation.*

## 6.4 Additive and multiplicative limitations

This section further investigates the two limitation techniques adopted in this chapter to comply with stability condition, i.e., $\lambda < \mu$. To avoid $\lambda = \mu$, an additive limitation and a multiplicative one are used. In the former case, we consider only the values of $\Lambda$ and $M$ such that $\lambda \leq \mu - \epsilon$, where $\epsilon$ is an arbitrarily small positive quantity. In the latter one, the values of $\Lambda$ and $M$ are those for which $\lambda \leq \mu/k$, where $k$ is an arbitrary quantity larger than 1.

In Figure 6.7 the two kinds of limitations are compared with respect to the pairs of inter-arrival and service rate samples distributed with three different Erlang distributions represented by the ellipses. In fact, each ellipse depicted in Figure 6.7 is defined as:

$$\left(\frac{(\lambda - \hat{\lambda})\sqrt{n}}{c \cdot \hat{\lambda}}\right)^2 + \left(\frac{(\mu - \hat{\mu})\sqrt{m}}{c \cdot \hat{\mu}}\right)^2 = 1 \tag{6.29}$$

where $\hat{\lambda}$ and $\hat{\mu}$ are the average values of inter-arrival and service rates, respectively, $n$ and $m$ are the number of samples collected for the two estimates and $c$ is a constant factor that multiplies $\hat{\theta}/\sqrt{l}$ (with $\hat{\theta} = \{\hat{\lambda}, \hat{\mu}\}$ and $l = \{m, n\}$), the standard deviation of an $l$-stage Erlang distribution with rate $l/\hat{\theta}$. As shown in Figure 6.7, where $\epsilon = 1$ and $k = 2$, for some distribution of $\Lambda$ and $M$ (e.g., those represented by ellipse $A$) $k$-limitation provides better results since it considers a larger area than the $\epsilon$-one. $\epsilon$-limitation works better than $k$-limitation for other distributions of the two rates (e.g., ellipse $C$). Finally, there are distributions of inter-arrival and service rates for which both the limitations can be used with no distinction (e.g., ellipse $B$).

The efficiency of both the limitations depends on the values of several parameters: $\epsilon$ (or $k$), $\hat{\lambda}$, $\hat{\mu}$, $m$ and $n$. Furthermore, limited to the Erlang

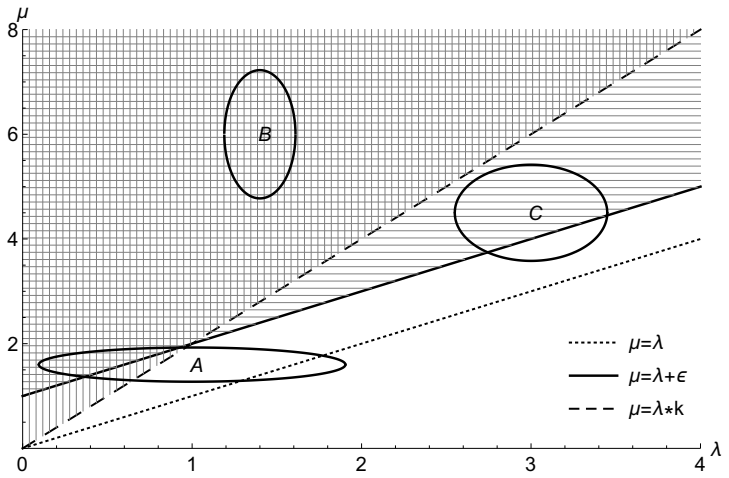


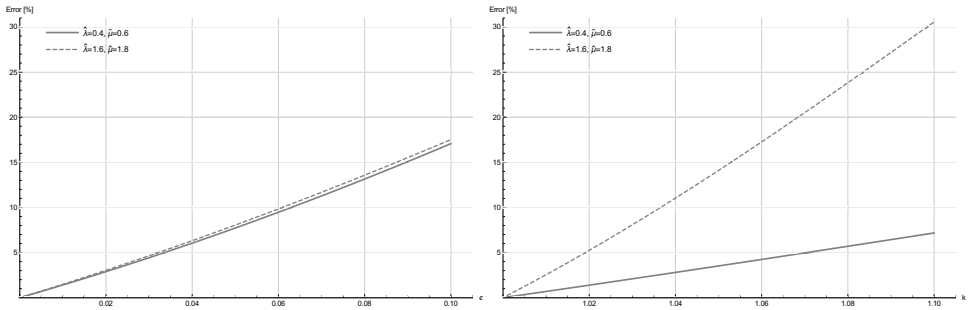

**Figure 6.7:** *Comparison of $\epsilon$-limitation and $k$-limitation for different Erlang distributions.*

case, the area taken into account by each limitation is not enough to evaluate its efficiency, since the analyzed metric (e.g., average number of entities in the system, system response time, etc.) also affects the result. For these reasons, the selection of a limitation is not an easy task that can be completed a priori. In fact, the results shown in Section 6.3 have been derived and studied for both the limitations.

In order to determine the accuracy of the two limitations, their error may be evaluated as follows:

$$Error = \frac{|A_{tot} - A_{lim}|}{A_{tot}} \tag{6.30}$$

where $A_{tot}$ is the area considered when limitations are not used (i.e., $\lambda = \mu$ are legit values) and $A_{lim}$ is the area considered when $lim = \{\epsilon, k\}$ limitation is adopted. Since no metrics are taken into account, Eq. (6.30) evaluates the accuracy of the two limitations considering how many combination of samples are discarded and not used for the evaluation of the output measure. For example, Figure 6.8 shows the errors made by $\epsilon$ and $k$ limitations as a function of $\epsilon$ and $k$, respectively. The results presented in the two graphs have been obtained considering two different cases. The straight lines have been computed with $\Lambda \sim Erlang(7, 17.5)$ and $M \sim Erlang(16, 26.67)$, whereas for the dashed lines $\Lambda \sim Erlang(110, 68.75)$ and $M \sim Erlang(139, 77.22)$ have been adopted. In both cases, the error made by the limitations increases with the absolute value of $\epsilon$ and $k$ since a larger area is discarded. Furthermore, as shown in Figure 6.7, the performance of $k$-limitation is also affected by the values of $\hat{\lambda}$ and $\hat{\mu}$ and their




**(a)** **(b)**

**Figure 6.8:** *Error for Erlang distributed* $\Lambda$ *and* $M$*, as a function of (a)* $\epsilon$ *and (b)* $k$*.*

ratio. Indeed, the area taken into consideration by this limitation decreases when $\hat{\lambda}/\hat{\mu}$ is close to 1 and the values of the two input parameters are large.

CHAPTER 7

# Epistemic Uncertainty Propagation in Power Models

## 7.1 Motivation

As happens for many models, also those used to estimate systems' power consumption are usually solved assuming fixed values for their input parameters. However, since those parameters are estimated from a finite number of observations, they are not exactly known and should be expressed as confidence intervals. Thus, they introduce epistemic uncertainty into the model due to partial information and lack of knowledge. As stated in Chapter 2 and 6, epistemic uncertainty must be propagated through the model in order to determine how it is affecting the model's output measures.

In this chapter, epistemic uncertainty is propagated through a power consumption model, when the data-center arrivals forms a homogeneous Poisson process and service times are exponentially distributed (i.e., the rates are fixed over time).

Epistemic uncertainty propagation is applied to a power model for two main reasons: *i)* to investigate how the input parameters affect the output measure of the power model (i.e., power consumption estimates); *ii)* to determine the accuracy of each input parameter required to estimate the

power consumption with a given confidence.

## 7.2 Modeling M/M/c/K queues power consumption

The non-linear power model presented in [49] and defined by Eq. (2.8) is the starting point of this chapter. *Stress*[8] is the benchmark used to generate the CPU workload and *Machine2* described in Chapter 3 is the system taken into consideration. In order to analyze different CPU utilization (i.e., $U = \{0\%,\ 25\%,\ 50\%,\ 75\%,\ 100\%\}$), *Stress* works with a different amount of threads (i.e., *Threads* $= \{0,\ 1,\ 2,\ 3,\ 4\}$) and the SMT level of each core is set to 1. Note that, power consumption samples observed when $U = 0\%$ and $U = 100\%$ are $P_{idle}$ and $P_{busy}$ samples, respectively.

Before adopting Eq. (2.8) to study the power consumption of a server, samples collection and parameters estimation are required. As previously said, the finite number of observations for input parameters is the source of epistemic uncertainty. Thus, in this chapter, uncertainty is introduced into the system by the stochastic parameters $P_{idle}$, $P_{busy}$, $r$ and $U$.

In order to decrease the number of random variables, $P_{busy}$ is expressed as a function of $P_{idle}$. To this end, samples analysis and algebraic manipulation have been performed and a linear relationship between the two parameters has been identified. The relationship is defined by the equation $P_{busy} = m \cdot P_{idle} + q$ with $m = 0.796211$ and $q = 61.8296$, in the case of the architecture considered (i.e., *Machine2*). In Figure 7.1 a Q-Q plot shows the existing linear relationship between $P_{busy}$ and $P_{idle}$. Some points are not fitted by the provided linear equation; that is due to some background processes that were executed during the measurement campaign, making the system collect some $P_{idle}$ samples when $U \neq 0\%$.

Thus, power model given in Eq. (2.8) is extended considering the linear relationship, and it becomes:

$$P(U, P_{idle}, r) = P_{idle} + [q + (m-1) \cdot P_{idle}] \cdot (2U - U^r) \tag{7.1}$$

with $m = 0.796211$ and $q = 61.8296$.

The effectiveness of the power model defined by Eq. (7.1) is shown in Figure 7.2, where a Q-Q plot compares $[P_{Est}|U]$, the distribution of the system power consumption estimated through Eq. (7.1) with a fixed value $U$, with $P_{Samp}(U)$, the power consumption distribution estimated from the samples collected during the measurement campaign for the considered utilization level $U$. In particular, each realization of $[P_{Est}|U]$ is determined starting from the sampled distribution of the idle system, $P_{Samp}(0)$, the

[8] `https://people.seas.harvard.edu/~apw/stress/`. Accessed: Jan. 15, 2018.

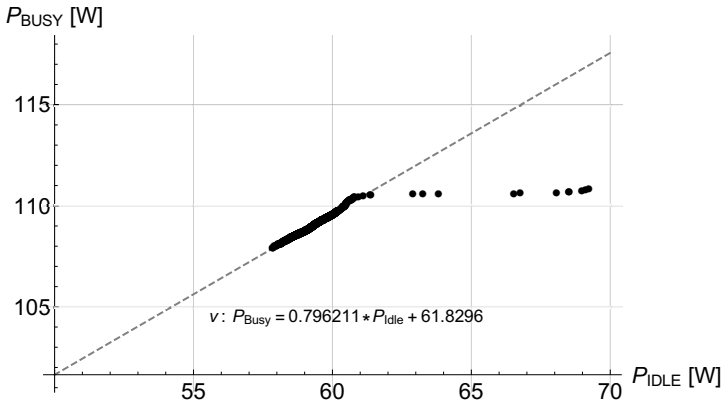



**Figure 7.1:** *Q-Q plot representing the linear relationship between $P_{idle}$ and $P_{busy}$ samples.*

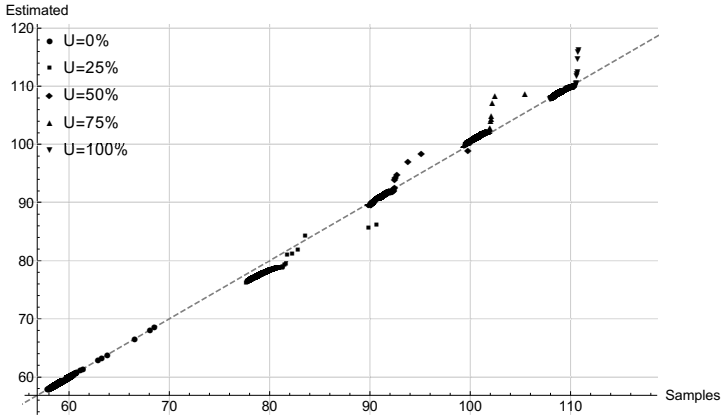



**Figure 7.2:** *Q-Q plot comparing the distribution of the estimated power consumption $[P_{Est}|U]$ against the sampled one $P_{Samp}(U)$.*

considered utilization level $U$, and the calibration parameter $r = 1.45862$ (where such value has been estimated from the collected samples). In particular, we have:

$$[P_{Est}|U] = P_{Samp}(0) + [61.8296 - 0.203789 \cdot P_{Samp}(0)] \cdot (2U - U^{1.45862}) \tag{7.2}$$

As it can be seen, the obtained curves are closely aligned to the $45^\circ$ line for all values of $U$, showing that the proposed procedure is capable of estimating very well the real power consumption distribution for a given utilization level $U$.

In order to avoid the stability issues, an $M/M/c/K$ queue is adopted to model the server considered in this chapter. Thus, solving the proper

continuous time Markov chain (CTMC), $U$ is computed through equation [130]:

$$U = 1 - \sum_{i=0}^{c} \frac{(c-i)\cdot \pi_i}{c} \tag{7.3}$$

where the probability to be in state $i$ of the considered CTMC, $\pi_i$, is defined as:

$$\pi_i = \begin{cases} \frac{\Lambda^i}{i!M^i}\cdot \pi_0 & \text{for } 0 \leq i \leq c \\ \frac{\Lambda^i}{c^{i-c}c!M^i}\cdot \pi_0 & \text{for } c \leq i \leq K \end{cases} \tag{7.4}$$

and $\pi_0$, the probability to be in state $0$, is:

$$\pi_0 = \left( \sum_{i=0}^{c-1} \frac{\Lambda^i}{i!M^i} + \sum_{i=c}^{K} \frac{\Lambda^i}{c^{i-c}c!M^i} \right)^{-1} \tag{7.5}$$

Assuming to know the number of cores of the considered architecture, $c$, and its buffer capacity, $K$, the resource utilization is a stochastic object on the two r.v. $\Lambda$ and $M$, the arrival and service rates, respectively.

Deriving the utilization of the queue starting from $\Lambda$ and $M$ estimates, instead of collecting utilization samples, has a two-fold advantage: *i)* differently from $U$, estimation of $\Lambda$ and $M$ does not require to determine an optimal time window; *ii)* it enables *what-if* analysis, thus to study the system under different values of inter-arrival and service rates.

The densities of the input random variables (i.e., arrival rate $\Lambda$, service rate $M$, $P_{idle}$ and calibration parameter $r$) that may affect the output of the power model are analyzed in the following.

As said and proved in [97–99], the rate of an exponential density determined from a finite set of its samples is Erlang distributed. It is the case of the arrival and service rates when dealing with $M/M/c/K$ queues. In particular, since arrival (service) time, $X$, is exponentially distributed with rate $\lambda$ (resp. $\mu$), the random variable $S_X = \sum_{i=1}^{k_x} X_i$ follows a $k_X$-stage Erlang distribution with rate parameter $\lambda$ (resp. $\mu$), and its pdf, given $\Lambda = \lambda$ (resp. $M = \mu$), is:

$$\begin{aligned} f_{S_\Lambda|\Lambda}(s_\Lambda|\lambda) &= \frac{\lambda^{k_\Lambda} s_\Lambda^{k_\Lambda - 1} e^{-\lambda s_\Lambda}}{(k_\Lambda - 1)!}, \\ f_{S_M|M}(s_M|\mu) &= \frac{\mu^{k_M} s_M^{k_M - 1} e^{-\mu s_M}}{(k_M - 1)!} \end{aligned} \tag{7.6}$$

Applying Bayes theorem and using Jeffreys' prior for $\Lambda$ (resp. $M$) (i.e., $f_\Lambda(\lambda) = s_\Lambda/\lambda$, and $f_M(\mu) = s_M/\mu$) as done in [99], the pdf of rate param-

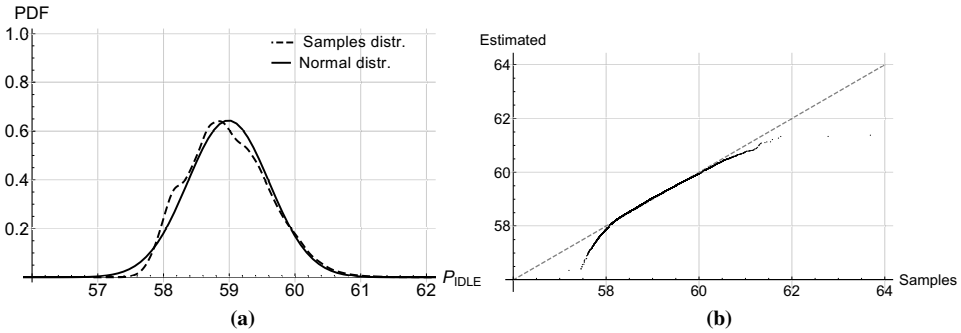



**Figure 7.3:** *On the left, the density of the collected $P_{idle}$ samples compared to a $Normal(58.992, 0.619657)$ one. On the right, a Q-Q plot that compares the collected samples to the random variates estimated through the Normal distribution.*

eter $\Lambda$ (resp. $M$) is derived as:

$$\begin{aligned} f_{\Lambda|S_\Lambda}(\lambda|s_\Lambda) &= \frac{\lambda^{k_\Lambda-1} s_\Lambda^{k_\Lambda} e^{-\lambda s_\Lambda}}{(k_\Lambda-1)!} = Erlang(k_\Lambda, s_\Lambda)\ , \\ f_{M|S_M}(\mu|s_M) &= \frac{\mu^{k_M-1} s_M^{k} e^{-\mu s_M}}{(k_M-1)!} = Erlang(k_M, s_M) \end{aligned} \tag{7.7}$$

With regard to $P_{idle}$ and calibration parameter $r$, Mathematica [74] has been used to fit the collected samples and find their probability densities. The results that estimate the $P_{idle}$ density are shown in Figure 7.3. They have been derived after collecting $20000$ samples. In particular, in Figure 7.3a the pdf of the collected samples is compared to the one of the estimated $Normal(58.992, 0.619657)$ density. In Figure 7.3b a Q-Q plot is used to analyze the two densities. The results confirm that the density of the $P_{idle}$ samples observed during the measurement campaign can be approximated by the estimated *Normal* density. The coefficient of variation ($c_v$) is computed as the ratio of the standard deviation to the mean, and for $P_{idle} \sim Normal(58.992, 0.619657)$ it is $c_v = 0.0105$. Since it is larger than $1\%$, the impact of $P_{idle}$ density on the output of the model described by Eq. (7.1) is further analyzed in the next Section.

In order to evaluate the calibration parameter $r$ density, $3700$ samples have been collected for each considered value of utilization, $U$. In fact, $3700$ samples has been found to be a good trade-off between the quality of the measure and the time required for samples collection. Then, $100$ sets have been created, each one composed by $185$ randomly selected samples (i.e., $37$ samples for each value of $U$). Finally, starting from the $100$ sets, Mathematica is used to estimate parameter $r$ density. The results are shown

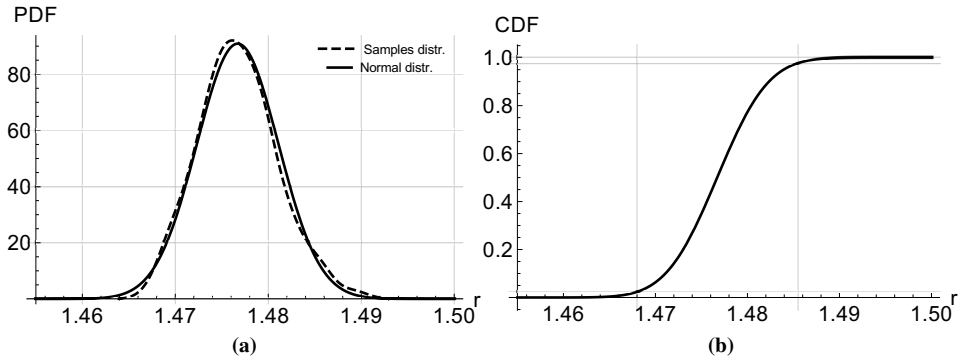



**Figure 7.4:** *On the left, the density of the estimated $r$ samples compared to a $Normal(1.45862, 0.00439)$ one. On the right, the Cdf of the normal distribution.*

in Figure 7.4, where the pdf and cumulative distribution function (Cdf) of $r \sim Normal(1.45862, 0.00439)$ is plotted. Due to its small coefficient of variation, $c_v = 0.003$, calibration parameter $r$ is not considered as a source of uncertainty for the purpose of this chapter and it is assumed to be exactly known.

## 7.3 Epistemic uncertainty in power models

In this Section, the techniques used to study uncertainty propagation in the power model described by Eq. (7.1) are presented. In order to identify the parameters that must be considered to obtain more accurate results, two different cases are analyzed: *i)* the uncertainty is introduced into the system by $P_{idle}$, $\Lambda$ and $M$, and *ii)* only $\Lambda$ and $M$ are the sources of uncertainty. Then, the validation of the chosen equation is performed, using Monte Carlo samples, for some different values of resource utilization, $U$.

### 7.3.1 Epistemic uncertainty in M/M/c/K queues

The equations for epistemic uncertainty propagation presented in Section 2.3.1 are now extended and adapted to the case of power consumption estimation in $M/M/c/K$ queues.

As said in Section 7.2, for the purpose of this chapter we assume epistemic uncertainty is introduced into the system by input random variables $P_{idle}$, $\Lambda$ and $M$, since $P_{busy}$ is expressed as a function of $P_{idle}$ and the density of the calibration parameter $r$ has a small coefficient of variation.
Considering three input parameters are introducing uncertainty into the power model, a three-dimensional integral (i.e., one for each input ran-

dom variable) is required to compute the unconditional Cdf and expected value of the output metric. In particular, assuming the epistemic random variables to be independent, Eq. (2.12) for deriving Cdf becomes:

$$F_{P(U|P_{idle}=p,\Lambda=\lambda,M=\mu)}(x) = \int_0^\infty \int_0^\infty \int_0^\infty \mathbb{1}(P(U|P_{idle}=p,\Lambda=\lambda,M=\mu) \leq x) \cdot f_\Lambda(\lambda) \cdot f_M(\mu) \cdot f_{P_{idle}}(p)\, d\lambda\, d\mu\, dp \tag{7.8}$$

and the one for expected value, i.e., Eq. (2.13), now is:

$$\mathbb{E}[P(U|P_{idle}=p,\Lambda=\lambda,M=\mu)] = \int_0^\infty \int_0^\infty \int_0^\infty P(U|P_{idle}=p,\Lambda=\lambda,M=\mu) \cdot f_\Lambda(\lambda) \cdot f_M(\mu) \cdot f_{P_{idle}}(p)\, d\lambda\, d\mu\, dp \tag{7.9}$$

The limits of integration of each input parameter vary between $0$ and infinity since there are not constraints on the values the input parameters may assume. In particular, since a finite capacity $M/M/c/K$ queue is considered, the system is always stable and no constraints on the value of $\Lambda$ are given. The epistemic densities of $\Lambda$ and $M$, i.e., $f_\Lambda(\lambda)$ and $f_M(\mu)$, are given in Eq. (7.7), whereas $f_{P_{idle}}(p)$ is the probability density function of the random variable $P_{idle}$, that is $P_{idle} \sim Normal(58.992, 0.619657)$ as shown in Section 7.2.

Let us call respectively $k_\lambda$ and $k_\mu$ the number of samples collected before estimating the values of $\Lambda$ and $M$. The Cdf of the power consumption and its expected value derived through Eqs. (7.8) and (7.9), respectively, are shown in Figure 7.5. They have been solved through numerical integrations – performed by Mathematica – due to the complexity to get closed form solutions. Both the statistical measures are plotted against the number of samples collected for $\Lambda$ and $M$ estimation. For the sake of simplicity, only the case when $k_\lambda = k_\mu$ (i.e., the same number of samples is collected to estimate arrival and service rates) is represented in Figure 7.5, and only $U \simeq 70\%$ is considered (i.e., average arrival rate $\hat{\lambda} = 6.4$ jobs/second, and average service rate $\hat{\mu} = 2.25$ jobs/second). As expected, the results show the importance to collect as many samples as possible for each input parameter to make their confidence interval tighter. Indeed, starting from the number of collected samples, $k$, it is possible to derive the confidence interval of an exponential distributed random variable as follows [97]:

$$2d = \frac{1}{2s_k} \left\{ \chi^2_{2k,\alpha/2} - \chi^2_{2k,1-\alpha/2} \right\} \tag{7.10}$$

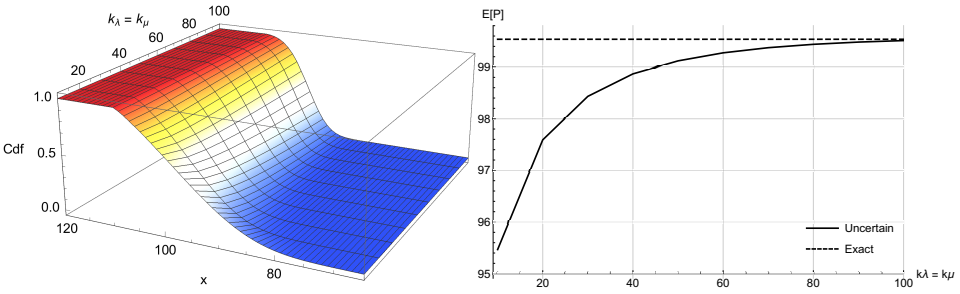



**Figure 7.5:** *The power consumption Cdf (a) and expected value (b) plotted as a function of the collected samples for the two input parameters, when* $\hat{\lambda} = 6.4$ *j/s and* $\hat{\mu} = 2.25$ *j/s.*

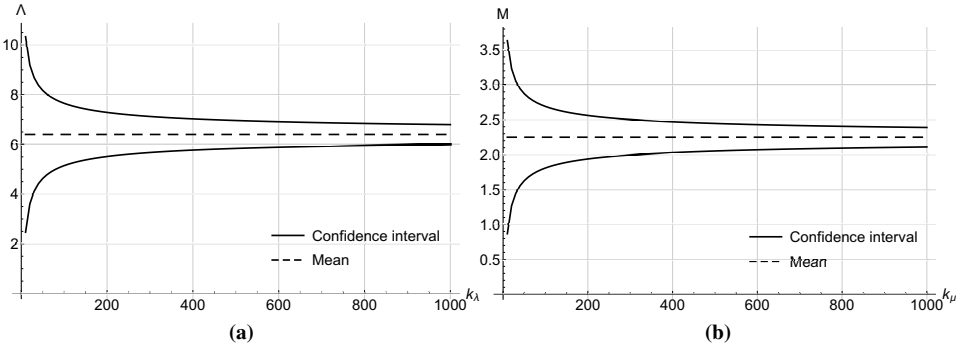



**Figure 7.6:** $95\%$ *confidence interval for the power model input parameters* $\Lambda$ *and* $M$ *as a function of the number of observations. On the left, the confidence interval length for* $\hat{\lambda} = 6.4$ *j/s; on the right, the confidence interval length for* $\hat{\mu} = 2.25$ *j/s.*

where $d$ is the half-width of the confidence interval of the considered input random variable, $s_k$ is the sum of the $k$ collected samples and $\chi^2_{2k,1-\alpha/2}$ is the critical value of the chi-square distribution with $2k$ degrees of freedom. The confidence interval length of the random variables $\Lambda$ and $M$, computed with Eq. (7.10) for $U \simeq 70\%$, are plotted in Figure 7.6 as a function of the number of observations. The straight lines are the lower and upper bounds, while the dashed ones are the average values $\hat{\lambda} = 6.4$ j/s and $\hat{\mu} = 2.25$ j/s. Similar results may be obtained for different resource utilization, varying the average arrival rate, $\hat{\lambda}$.

In order to evaluate the impact of the input random variable $P_{idle}$ on the output metric of the considered power model, Eqs. (7.8) and (7.9) are adapted to make them take into account only the effect of $\Lambda$ and $M$, assum-

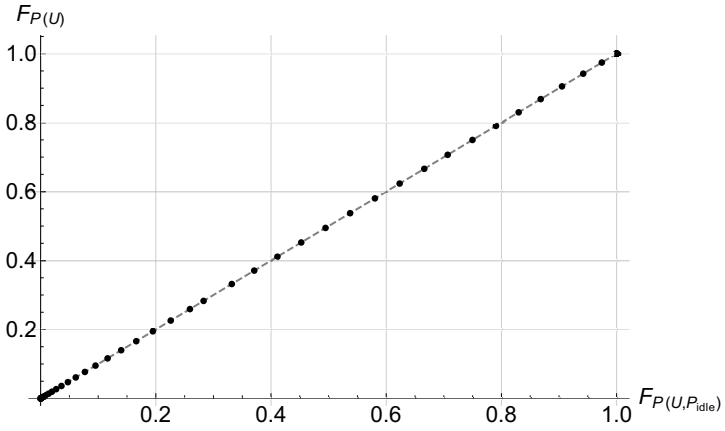



**Figure 7.7:** *P-P plot that compares the power consumption Cdfs computed through Eqs. (7.8) and (7.11), when $k_\lambda = k_\mu = 10$.*

ing $P_{idle}$ is estimated with certainty. Thus, they become:

$$F_{P(U|\Lambda=\lambda,M=\mu)}(x) = \int_0^\infty \int_0^\infty \mathbb{1}(P(U|\Lambda=\lambda, M=\mu) \leq x) \cdot f_\Lambda(\lambda) \cdot f_M(\mu)\, d\lambda\, d\mu \tag{7.11}$$

and

$$\mathbb{E}[P(U|\Lambda=\lambda, M=\mu)] = \int_0^\infty \int_0^\infty P(U|\Lambda=\lambda, M=\mu) \cdot f_\Lambda(\lambda) \cdot f_M(\mu)\, d\lambda\, d\mu \tag{7.12}$$

respectively. Also in this case numerical solutions have been derived. Since the results computed through Eqs. (7.11) and (7.12) for $U \simeq 70\%$ are similar to those depicted in Figure 7.5, they are not plotted. In order to show that the Cdfs and expected values derived in the two different ways are identical, a P-P plot is used for the Cdfs comparison and is depicted in Figure 7.7. For the sake of space, only the P-P plot for $k_\lambda = k_\mu = 10$ is shown here. Similar results are obtained considering the Cdfs obtained with larger values of $k_\lambda = k_\mu$.

Based on the presented results, it is possible to assert that epistemic uncertainty introduced by $P_{idle}$ into the considered power model is negligible, and the uncertainty analysis can be performed just accounting for the densities of $\Lambda$ and $M$, thus adopting Eq. (7.11) to derive the Cdf of the power consumption, and Eq. (7.12) for its expected value. Note that this assumption has also the effect of reducing the computational complexity, allowing to deal with only a 2-dimensional integral.

Finally, in the limiting case $k_\lambda, k_\mu \to \infty$, the Erlang distributed input random variables have a zero variance, thus they are deterministic distributed (or degenerate) [70] and they take a single value (i.e., $\lambda = \hat{\lambda}$ and $\mu = \hat{\mu}$). In this case, the considered model is no more affected by epistemic uncertainty, and it may be studied without accounting for epistemic uncertainty propagation.

### 7.3.2 Validation

In order to validate Eq. (7.11), that is used to derive the Cdf of the power model given in Eq. (7.1) for $M/M/c/K$ queues, a further benchmark is run on the architecture presented in Chapter 3 (i.e., *Machine2*). The benchmark consists in executing a given number of CPU intensive requests. The arrival and service times for each request are passed by the user at the beginning of the test.

For the purpose of this validation, $1000$ exponentially distributed samples for arrival and service times are generated. This step is repeated for each value of resource utilization that has been studied. In particular, the average service rate $\hat{\mu}$ is set to $2.25$ jobs/second, and the average arrival rate varies among $\hat{\lambda} = \{0.8, 1.6, 2.4, 3.2, 4.0, 4.8, 5.6, 6.4, 7.2\}$ jobs/second, in order to consider different resource utilization, that are derived as shown by Eq. (7.3).

In addition to the generated arrival and service times samples, the power consumption measures recorded by a Yokogawa WT210 power meter are also collected.

For the sake of clarity, only the results for $U \simeq 9\%$ and $U \simeq 70\%$ (i.e., $\hat{\lambda} = 0.8$ and $\hat{\lambda} = 6.4$, respectively) are depicted in Figure 7.8. They are plotted as a function of the number of samples observed for arrival and service rates estimation, $k_\lambda$ and $k_\mu$, that in this chapter are assumed to be always equal.

In both the figures, the lower and upper bounds (i.e., the straight lines) have been derived starting from the power consumption Cdf computed through Eq. (7.11). Then, the $95\%$ confidence interval has been derived for each Cdf, and the two bounds have been plotted varying the number of observations.

The average power consumption measured by means of the power meter and independent on the number of collected samples is depicted in Figures 7.8a and 7.8b as a dotted line.

The samples of arrival and service times are used to plot the estimated power consumption (one point for each estimate). For this purpose, $k_\lambda$

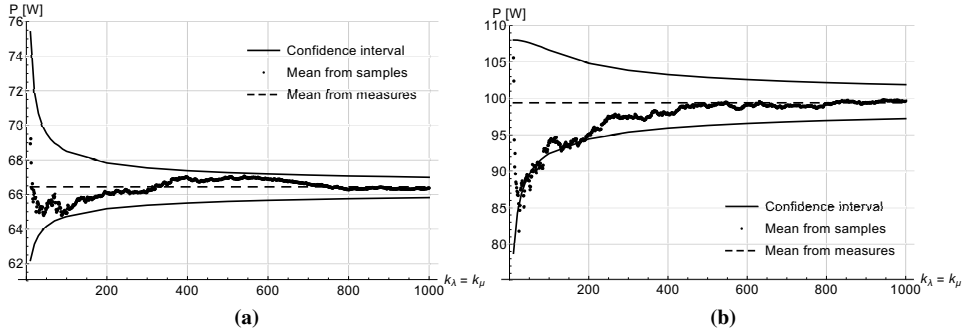



**Figure 7.8:** *The results for $U \simeq 9\%$ are depicted on the left, those for $U \simeq 70\%$ are depicted on the right.*

and $k_\mu$ samples (i.e., the number of generated Monte Carlo samples) are averaged in order to derive the mean arrival and service rates (i.e., $\hat{\lambda}$ and $\hat{\mu}$), respectively. Then, after using these values to compute $U$ as in Eq. (7.3), it is possible to estimate average power consumption through Eq. (7.1), with $P_{idle}$ and $r$ set to their average value. The larger the number of observations of arrival and service times, the more accurate the power consumption estimates.

As expected, a great amount of power consumption estimates lies between the two bounds and they are closer to the mean value when a large amount of samples is observed. While in Figure 7.8a few samples for both the parameters are enough to obtain values close to the average one, when the utilization of the system is $U \simeq 70\%$ (Figure 7.8b) at least 300 observations are required for both $\Lambda$ and $M$ before obtaining some accurate power consumption estimates.

## 7.4 Exploitation: life of a UPS battery

In this section the previous considerations about epistemic uncertainty propagation in power models are applied to a real exploitation case. In fact, a UPS battery that starts working after the power source failed is analyzed. In particular, the battery lifetime of a UPS, that must keep on a server which is processing a constant workload, is estimated.

Three main UPS architectures exist [145]: *i)* on-line UPS, that provides electrical power from the battery whether there is a failure or not; *ii)* standby UPS, that provides electrical power from the battery only if power source failed; *iii)* line-interactive UPS, that are high-class standby UPS. For the purpose of this section, distinction between the three architectures

is irrelevant since, in the scenario considered, the power source has already failed.

Several models have been proposed in literature in order to estimate a UPS battery lifetime: some of them, such as the electro-chemical models [44], are very accurate and need several input parameters to work; some others, like the kinetic battery model (KiBaM) [71, 77], need just a few parameters and may be easily used also by inexperienced users. In this section, the latter model is adopted.

The KiBaM is an accurate and simple model [71, 77] that divides the stored charge into two different wells: the *available charge* and the *bound charge*. The two wells are connected by a pipe and the charge can migrate in both directions – based on the level of charge in the two wells – at a given rate $\omega$. At the beginning, a fraction $\phi = [0, 1]$ of the total capacity is in the available charge well, and the remaining $1 - \phi$ fraction is in the bound charge one. When the battery is strained, only the available charge may be used, whereas the bound charge slowly becomes available charge. For the purpose of this section, the battery lifetime equation is derived as in [77] by solving the differential equations that model the dynamic of the charge of both the wells that, when the load is assumed to be constant, is:

$$I \cdot t + \frac{(1-\phi) \cdot I}{\phi} \cdot \frac{1 - e^{-\omega' t}}{\omega'} = \hat{\gamma} \tag{7.13}$$

where $I$ is the load that is straining the battery, $t$ is the lifetime of the battery, $\hat{\gamma}$ is the battery full capacity and $\omega' = \omega/[\phi(1-\phi)]$.

Although [71] has already analyzed how uncertainty on input parameters (i.e., initial capacity and load) affects the KiBaM performance in predicting the GomX-1[9] satellite state of charge, they expressed the uncertainty on the output measure without considering the number of observations of the input parameters. In this sense, our technique lets the users consider the amount of samples to be collected in order to get a more accurate output measure.

Note also that the batteries are usually not fully drained in order to prevent damages [34]. Thus, a UPS unit is shut down when the battery capacity is lower than a fixed threshold. Nevertheless, in this chapter it is assumed the battery can completely drain and the UPS unit does not shut down until the battery is exhausted. Note that, the standard UPS battery configuration may be considered after some adjustments to Eq. (7.13).

Battery lifetime is derived solving the kinetic battery model for fixed load given in Eq. (7.13). For this purpose, $\phi = 1/2$ and $\omega = 1/100$ as done by Hermanns et al. in [71]. They also assumed the battery initial capacity

[9] `https://gomspace.com/gomx-1.aspx`. Accessed: Jan. 15, 2018.

is uniform distributed between $70\%$ and $90\%$ of full capacity $\hat{\gamma}$, thus assuming to deal with a random environment. Furthermore, in our case, load $I$ is affected by epistemic uncertainty due to the finite number of samples collected for the input parameters $\Lambda$ and $M$ (i.e., arrival and service rates, respectively). Indeed, the current load $I$ is derived starting from server utilization in Eq. (7.3); utilization is used to compute the server's power consumption through Eq. (7.1) that is divided by voltage $\Delta V$ to obtain the load $I$ (i.e., $I = P(U, P_{idle}, r)/\Delta V$). This is a different assumption with respect to [71], where also the load was affected by aleatory uncertainty. Finally, voltage is $\Delta V = 12$ V, and the capacity is $\hat{\gamma} = 9$ Ah; both of them are from the specifications of Trust OXXTRON 1000VA UPS.

Starting from Eq. (7.13), battery lifetime $L$ is defined as:

$$L(\lambda, \mu, \gamma) = t \mid I(\lambda, \mu) \cdot t + \frac{(1-\phi) \cdot I(\lambda, \mu)}{\phi} \cdot \frac{1 - e^{-\omega' t}}{\omega'} = \gamma$$

For the sake of simplicity, the dependency of $L$ on $\lambda$, $\mu$ and $\gamma$ has been dropped. Thus, the epistemic uncertainty of input parameters $\Lambda$ and $M$ is propagated to the battery lifetime as follows:

$$F_L(l) = \int_0^\infty \int_0^\infty \int_0^\infty \mathbb{1}(L \leq l) \cdot f_\Lambda(\lambda) \cdot f_M(\mu) \cdot f_\Gamma(\gamma)\, d\lambda\, d\mu\, d\gamma \quad (7.14)$$

where

$$f_\Gamma(\gamma) \sim Uniform\left(\frac{70\hat{\gamma}}{100}, \frac{90\hat{\gamma}}{100}\right),$$

whereas $f_\Lambda(\lambda)$ and $f_M(\mu)$ are the Erlang densities given in Eq. (7.7).

The Cdfs obtained solving Eq. (7.14) for $\hat{\lambda} = 0.8$ j/s and $\hat{\lambda} = 6.4$ j/s are depicted in Figures 7.9a and 7.9b, respectively. Although tighter confidence intervals are derived increasing the number of observation for arrival and service rates from $10$ to $100$, a smaller improvement is obtained when the number of collected samples is further increased. Indeed, in both cases depicted in Figure 7.9, the Cdf of the battery lifetime for $k_\lambda = k_\mu = 1000$ is very similar to the one derived with $k_\lambda = k_\mu = 100$.

In fact, for the case-study considered, the inaccuracy of output measure is mainly due to the aleatory uncertainty introduced by the unknown battery capacity $\gamma$. That may also be seen considering the limiting case for $k_\lambda, k_\mu \to \infty$ discussed in Section 7.3, when arrival and service rates, $\Lambda$ and $M$, follows a degenerate distribution (i.e., they are known with certainty), while the full capacity $\gamma$ is still uniformly distributed. The battery lifetime Cdf for $k_\lambda = k_\mu = 1000$ and $k_\lambda, k_\mu \to \infty$, when $\lambda = 6.4$ j/s, are plotted in

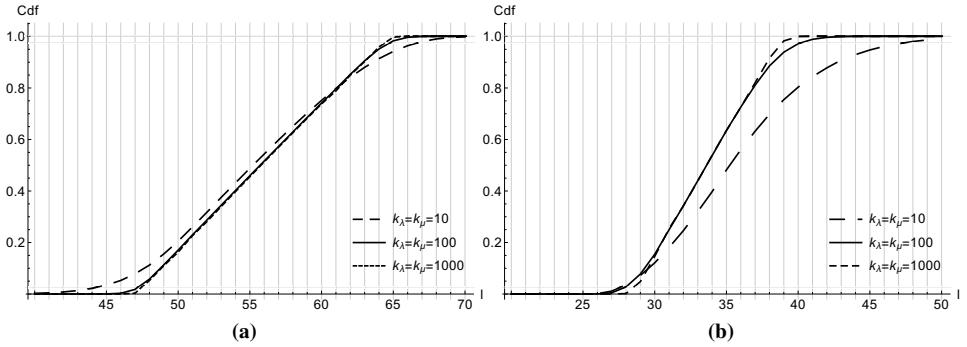



**Figure 7.9:** *Battery lifetime Cdfs when the resource utilization is* $U \simeq 9\%$ *(on the left) and* $U \simeq 70\%$ *(on the right).*

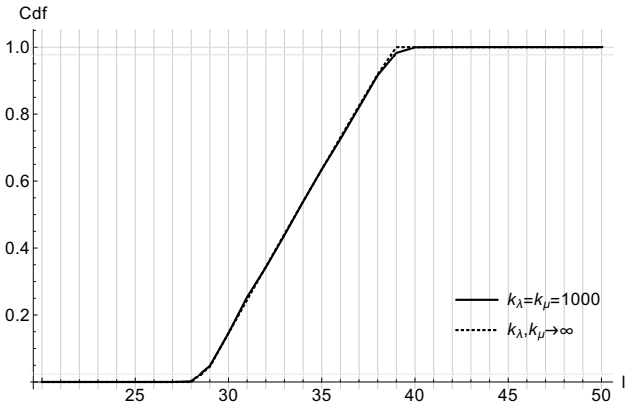



**Figure 7.10:** *Battery lifetime Cdfs when the resource utilization is* $U \simeq 70\%$ *and the number of collected samples is* $1000$ *or tends to infinity.*

Figure 7.10, where it is shown that the difference between the two Cdfs is negligible. Similar results are obtained for $\lambda = 0.8$ j/s.

CHAPTER 8

# Parametric sensitivity to study epistemic uncertainty propagation

## 8.1 Motivation

Epistemic uncertainty is propagated into a model to study the effects of inaccurate input parameters on the output measures, thus getting results with confidence intervals. As already presented, the technique adopted in this thesis for epistemic uncertainty propagation is based on multi-dimensional integrals. Unfortunately, it is not always easy to study the uncertainty propagation with that technique. Indeed, for large and complex models, it is often not practical to analytically or numerically compute the uncertainty through integrals.

Parametric sensitivity analysis allows the modeler to identify the input parameters that most affect the output measures of the model. Differently from epistemic uncertainty propagation, parametric sensitivity can be easily computed in short time also considering complex models.

The purpose of this chapter is to find a relationship between epistemic uncertainty and parametric sensitivity, in order to enable a faster uncertainty analysis also on large models with several input parameters.

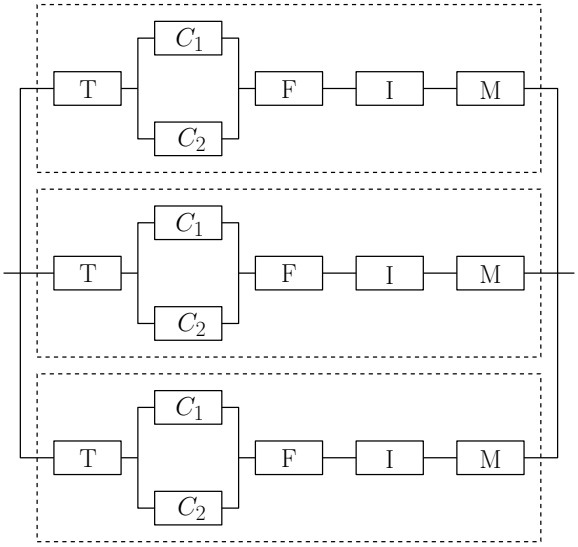



**Figure 8.1:** *Reliability block diagram of the multi-voltage propulsion system.*

## 8.2 The system and its model

The multi-voltage propulsion system for the Italian High Speed Railway [33] is the case-study used in this chapter. In particular, the dependability of this system is here considered.

Although in this chapter the epistemic uncertainty propagation and parametric sensitivity analyses are applied to a dependability model, the results obtained can be also adopted when studying different type of models (e.g., performance models).

The multi-voltage propulsion system, originally proposed in [33] and [37], consists of three equivalent modules in a parallel redundant configuration. The reliability block diagram of the system is shown in Figure 8.1. Each module is composed of transformer ($T$), filter ($F$), inverter ($I$) and motor ($M$) in series with two parallel converters ($C_1$ and $C_2$). In [33], the components' failure times are assumed to be exponentially distributed, and their failure rates are reported in Table 8.1.

Denote by $\lambda = \lambda_T + \lambda_F + \lambda_I + \lambda_M$ the total failure rate of the series components and $\gamma$ the failure rate of each parallel component. The propulsion system works until one of the three parallel modules is up.

A single module of the multi-voltage propulsion system is modeled by the CTMC in Figure 8.2. In *State 2* all the components are working and the module is fully operational. In *State 1* the module is still working, but only one of the two converters is operative, whereas *State 0* is the down state and

**Table 8.1:** *Failure rates of the system's components.*

| Component | Failure rate [f/h] |
|---|---|
| Transformer | $\lambda_T = 2.2 \cdot 10^{-6}$ |
| Converter | $\gamma = 2.8 \cdot 10^{-5}$ |
| Filter | $\lambda_F = 4.0 \cdot 10^{-7}$ |
| Inverter | $\lambda_I = 3.8 \cdot 10^{-5}$ |
| Motor | $\lambda_M = 3.2 \cdot 10^{-5}$ |

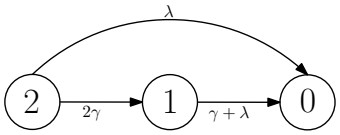



**Figure 8.2:** *Continuous time Markov chain model for one module of the multi-voltage propulsion system.*

the module is broken. The CTMC goes from *State 2* to *State 1* when one of the two converters fails. If a series component breaks up, the CTMC goes to *State 0* independently of its current state. Note that, if the CTMC is in *State 1*, the working converter is considered as a series component.

The fully symbolic transient solution of the CTMC in Figure 8.2 may be derived resorting to Laplace transforms, and it is:

$$\begin{cases} \pi_2(t) = e^{-(2\gamma+\lambda)t} \\ \pi_1(t) = 2e^{-(\gamma+\lambda)t} - 2e^{-(2\gamma+\lambda)t} \\ \pi_0(t) = 1 - \pi_1(t) - \pi_2(t) \end{cases} \tag{8.1}$$

Considering only the working states, the reliability of the module is derived as:

$$R(t) = \pi_2(t) + \pi_1(t) = 2e^{-(\gamma+\lambda)t} - e^{-(2\gamma+\lambda)t} \tag{8.2}$$

and its mean time to failure (MTTF) is:

$$MTTF = \int_0^\infty R(t)dt = \frac{3\gamma + \lambda}{(2\gamma + \lambda)(\gamma + \lambda)} \tag{8.3}$$

that, after substituting $\lambda$ and $\gamma$ with the values in Table 8.1, is $MTTF = 12104.7$ hours.

In order to consider the whole system in Figure 8.1, we assume statistical independence among the three parallel modules and use a hierarchical approach whose structure is depicted in Figure 8.3.

The highest level of the hierarchy is implemented with a multivalued Fault tree, in which the top event is the system failure; it is the output of

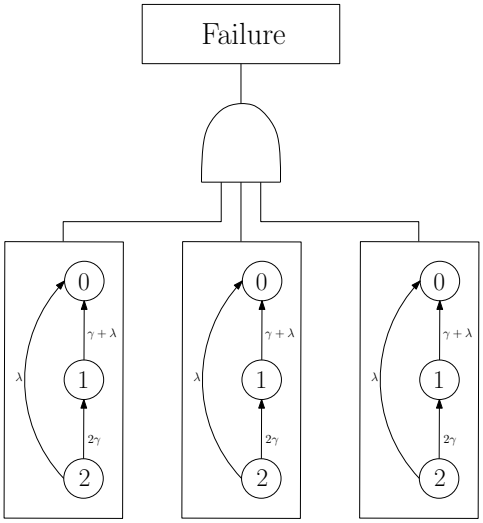



**Figure 8.3:** *Hierarchical model of the multi-voltage system.*

an AND gate whose inputs are the module failures. At the lower level of the hierarchy there are the CTMCs of Figure 8.2 modeling each module's failures. The top event occurs when all the inputs occur, i.e., all the three modules have failed.

Each module can be in one of the three $i$ states of Figure 8.2 (i.e., $i = \{0,\ 1,\ 2\}$), and the total number of modules in state $i$ is referred to as $n_i$, where $0 \leq n_0, n_1, n_2 \leq 3$ and $n_0 + n_1 + n_2 = 3$. Hence, the state of the top event can be described by a triple $\{n_0, n_1, n_2\}$ indicating how many modules are in state 0, in state 1 and in state 2, respectively.

The probability of each possible output configuration is derived as:

$$\begin{aligned} \pi_{n_0,n_1,n_2} &= \binom{3}{n_0, n_1, n_2} \cdot [\pi_2]^{n_2} \cdot [\pi_1]^{n_1} \cdot [\pi_0]^{n_0} \\ &= \frac{3!}{n_0!\, n_1!\, n_2!} \cdot [\pi_2]^{n_2} \cdot [\pi_1]^{n_1} \cdot [\pi_0]^{n_0} \end{aligned} \tag{8.4}$$

where the $\pi_i$ in Eq. (8.4) have been derived in Eq. (8.1).

Given the hierarchical model of the multi-voltage propulsion system, its reliability is computed as

$$R_{sys}(t) = 1 - \pi_{3,0,0}$$

where $\pi_{3,0,0}$ is the down state and is obtained from Eq. (8.4). Solving the

previous equation, the reliability of the multi-voltage propulsion system is:

$$\begin{aligned} R_{sys}(t) &= 1 - [1 - 2e^{-(\gamma+\lambda)t} + e^{-(2\gamma+\lambda)t}]^3 \\ &= 12e^{-(3\gamma+2\lambda)t} + 6e^{-(5\gamma+3\lambda)t} - 12e^{-(4\gamma+3\lambda)t} \\ &\quad - 3e^{-(2\gamma+\lambda)t} - 3e^{-2(2\gamma+\lambda)t} - e^{-3(2\gamma+\lambda)t} \\ &\quad + 6e^{-(\gamma+\lambda)t} - 12e^{-2(\gamma+\lambda)t} + 8e^{-3(\gamma+\lambda)t} \end{aligned} \tag{8.5}$$

Thus, the MTTF of the whole system is computed integrating its reliability as follows:

$$\begin{aligned} MTTF &= \int_0^\infty R_{sys}(t)dt \\ &= \frac{8}{3(\gamma+\lambda)} - \frac{29}{6(2\gamma+\lambda)} \\ &\quad + \frac{12}{(3\gamma+2\lambda)} - \frac{12}{(4\gamma+3\lambda)} + \frac{6}{(5\gamma+3\lambda)} \end{aligned} \tag{8.6}$$

and substituting $\lambda$ and $\gamma$ to the values given in Table 8.1, we obtain $MTTF = 21662.9$ hours.

## 8.3 Parametric sensitivity for epistemic uncertainty propagation

In this section parametric sensitivity is considered in order to find a fast strategy for epistemic uncertainty propagation. For this purpose, the input parameters that most affect a chosen output measure (in this case the MTTF of the system) are identified through parametric sensitivity, then epistemic uncertainty propagation is used to study how lack of knowledge on input parameters is affecting the chosen metric. Finally, a possible relationship between parametric sensitivity and epistemic uncertainty is investigated.

### 8.3.1 Parametric sensitivity

Parametric sensitivity is a technique used to identify the components that most affect an output measure. In this chapter we focus on the MTTF whose expression is given in Eq. (8.6).

Differential analysis [66, 91] is the basis of many sensitivity analysis techniques. The MTTF sensitivity with respect to a generic input parameter $\theta$, referred to as $S_\theta(MTTF)$, is obtained by computing the partial derivative of the metric of interest with respect to the parameter considered

as follows:

$$S_\theta(MTTF) = \frac{\partial MTTF}{\partial \theta}, \quad \theta = \{\lambda_T, \lambda_F, \lambda_I, \lambda_M, \gamma\} \tag{8.7}$$

The dimensionless *scaled sensitivity* is defined for each input parameter $\theta$ as [51]:

$$SS_\theta(MTTF) = \frac{\partial MTTF}{\partial \theta} \cdot \frac{\theta}{MTTF} \tag{8.8}$$

Adoption of scaled or unscaled sensitivity depends on several factors (e.g., type of the output metrics, the range of values of the input and output parameters, etc.). In this chapter we use the *scaled sensitivity* as in Eq. (8.8).

For each series component with failure rate $\lambda_X$, where $X = \{T, I, F, M\}$, the scaled sensitivity in Eq. (8.8) becomes:

$$SS_{\lambda_X}(MTTF) = \\ -\lambda_X \cdot \frac{\frac{8}{3(\gamma+\lambda)^2} - \frac{29}{6(2\gamma+\lambda)^2} + \frac{24}{(3\gamma+2\lambda)^2} - \frac{36}{(4\gamma+3\lambda)^2} + \frac{18}{(5\gamma+3\lambda)^2}}{\frac{8}{3(\gamma+\lambda)} - \frac{29}{6(2\gamma+\lambda)} + \frac{12}{3\gamma+2\lambda} - \frac{12}{4\gamma+3\lambda} + \frac{6}{5\gamma+3\lambda}} \tag{8.9}$$

Instead, expanding Eq. (8.8) for the parallel converters with failure rate $\gamma$, we obtain:

$$SS_{\gamma}(MTTF) = \\ -\gamma \cdot \frac{\frac{8}{3(\gamma+\lambda)^2} - \frac{29}{3(2\gamma+\lambda)^2} + \frac{36}{(3\gamma+2\lambda)^2} - \frac{48}{(4\gamma+3\lambda)^2} + \frac{30}{(5\gamma+3\lambda)^2}}{\frac{8}{3(\gamma+\lambda)} - \frac{29}{6(2\gamma+\lambda)} + \frac{12}{3\gamma+2\lambda} - \frac{12}{4\gamma+3\lambda} + \frac{6}{5\gamma+3\lambda}} \tag{8.10}$$

In Table 8.2, the numerical values of MTTF scaled sensitivity computed through Eqs. (8.9) and (8.10) are reported. They have been sorted from the most to the least sensitive. Note that all the values are negative meaning that if the value of the parameter $\theta$ increases, the MTTF decreases. The component that most affect the MTTF is the inverter which is the one with the largest failure rate.

Figure 8.4 depicts the MTTF of the system as a function of the failure rate of a single component, thus assuming that the failure rates of all the other components are constant. In particular, the larger the absolute parametric sensitivity with respect to a component, the greater the gain that is obtained if the failure rate of that component is decreased. The straight gray line in Figure 8.4 is the MTTF of the whole system as derived through Eq. (8.6) with the failure rates given in Table 8.1.

**Table 8.2:** *Scaled sensitivity of the MTTF metric. The parameters are ordered from the one that most affect the output measure to the one to which MTTF is less sensitive.*

| $\theta$ | $SS_\theta(MTTF)$ |
|---|---|
| $\lambda_I$ | $-4.16997 \cdot 10^{-1}$ |
| $\lambda_M$ | $-3.51155 \cdot 10^{-1}$ |
| $\gamma$ | $-2.03316 \cdot 10^{-1}$ |
| $\lambda_T$ | $-2.41419 \cdot 10^{-2}$ |
| $\lambda_F$ | $-4.38944 \cdot 10^{-3}$ |

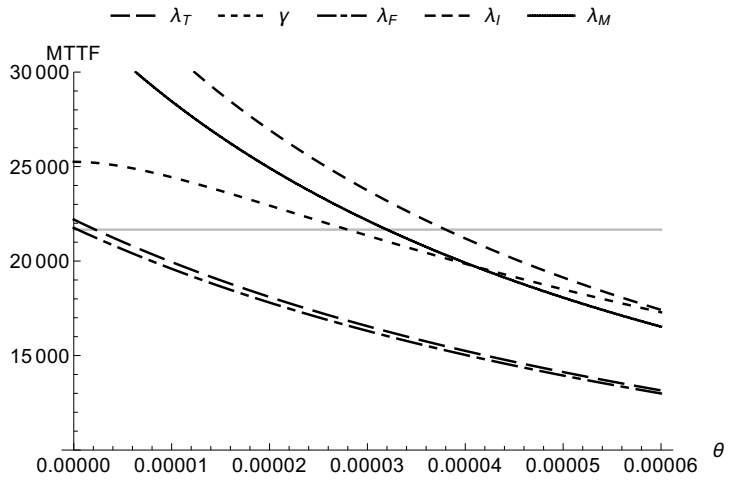



**Figure 8.4:** *MTTF of the system computed varying the failure rate of one component at a time. Each curve is obtained varying the considered parameter between* $0$ *and* $6 \cdot 10^{-5}$*, and keeping constant all the other failure rates. The straight line is the current value of MTTF, as computed from Eq. (8.6).*

### 8.3.2 Epistemic uncertainty

As said, the multi-dimensional integral technique proposed in [98, 125] and described in Section 2.3.1 is adopted to propagate epistemic uncertainty into the model.

However, in this chapter, only the variance of the output measure is taken into account since it is strictly related to the accuracy of the same metric. In fact, the smaller the variance, the tighter the confidence interval. In order to make the variance account for epistemic uncertainty, its definition is required:

$$Var[MTTF] = \mathbb{E}[MTTF^2] - (\mathbb{E}[MTTF])^2 \tag{8.11}$$

where the first and second moments are computed as in Eqs. (2.13) and (2.14), respectively, thus taking into consideration epistemic uncertainty.

As already shown in Chapters 6 and 7 and proved in [98], since the failure times of each component are exponentially distributed, the probability density function of each input random variable $\Theta$ is a $k$-stage Erlang distribution with rate parameter $s$:

$$f_{\Theta|S}(\theta|s) = \frac{\theta^{k-1} s^k e^{-\theta s}}{(k-1)!} \tag{8.12}$$

where $k$ is the number of collected samples and $s$ is their sum.

Figure 8.5 depicts the variance of the MTTF as a function of the number of samples collected for each input parameter, assuming the same number of samples is observed for all the input parameters. Note that, if the number of observations for each parameter tends to infinity (i.e., $k \to \infty$), thus the variance of the MTTF tends to zero. Indeed, in that case, the input parameters are exactly know and no epistemic uncertainty is introduced into the model.

The epistemic uncertainty of each input parameter $\theta$ may be evaluated assuming that all the input parameters are exactly known (i.e., infinite samples have been collected for their estimation), except the one of which we want to study uncertainty. This way, MTTF expected values is computed as:

$$\mathbb{E}[MTTF] = \int_0^\infty MTTF \cdot \frac{\theta^{k_\theta - 1} s_\theta^{k_\theta} e^{-\theta s_\theta}}{(k_\theta - 1)!} d\theta \tag{8.13}$$

and its variance can still be derived through Eq. (8.11) after computing its second moment as:

$$\mathbb{E}[MTTF^2] = \int_0^\infty MTTF^2 \cdot \frac{\theta^{k_\theta - 1} s_\theta^{k_\theta} e^{-\theta s_\theta}}{(k_\theta - 1)!} d\theta \tag{8.14}$$

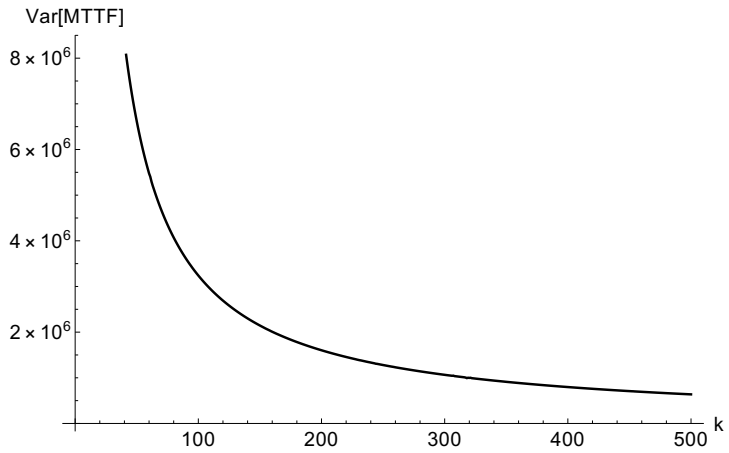



**Figure 8.5:** *Variance of the MTTF of the multi-voltage propulsion system as a function of the number of observations collected for each input parameter.*

Figure 8.6 depicts the MTTF variance as a function of the number of samples collected for the single parameter $\theta$ that is assumed to introduce epistemic uncertainty into the model. Note that, the more uncertain a parameter, the larger the amount of observations required for that parameter to make the MTTF variance close to zero. As in the previous case, the variance of the MTTF tends to zero when $k_\theta \to \infty$.

### 8.3.3 Parametric sensitivity vs. epistemic uncertainty

The closed-form approach to uncertainty propagation requires the computation of an $n$-dimensional integral, where $n$ depends on the number of input parameters. Both symbolic and numerical computations of such integral become impractical for complex and large models [97].

Moreover, that technique also requires to compute the joint epistemic density of all the input parameters. However, this task may be simplified by assuming that all the parameters are independent (as it has been done in this chapter and in the previous ones).

Sections 8.3.1 and 8.3.2 show that a relationship between parametric sensitivity and uncertainty propagation exists. Indeed, from Table 8.2 and Figure 8.6 it may be observed that for the input parameters with a larger absolute value of parametric sensitivity, also a larger amount of samples is required to estimate the output measure (i.e., MTTF) with a given confidence. The partial derivatives performed to derive parametric sensitivity are easier and quicker to be carried out than the multi-dimensional integral required to propagate epistemic uncertainty through a model.

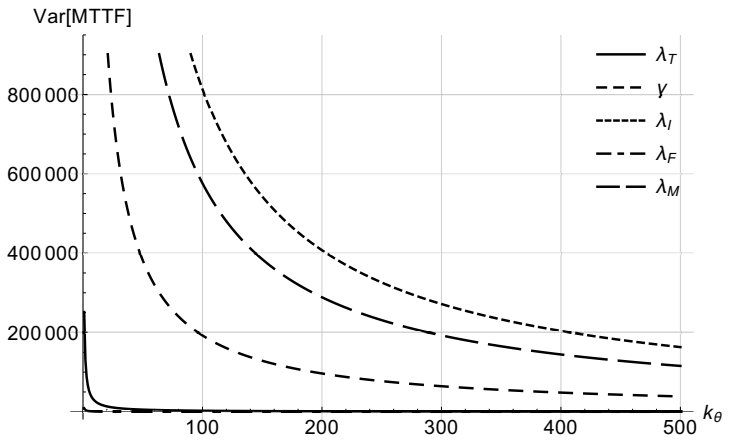



**Figure 8.6:** *Variance of the MTTF as a function of the number of samples collected for the single input parameter that is assumed to introduce uncertainty into the model.*

The above mentioned relationship allows us to use parametric sensitivity analysis to identify the parameters that need tighter confidence interval to get more accurate output measures. This method is less affected by the largeness and complexity of the system than the multi-dimensional integral technique.

However, the partial derivatives cannot predict how many samples are required to attain a given level of accuracy since the parametric sensitivity does not depend on the number of samples collected for the input parameters. For this reason, after using parametric sensitivity to identify the input parameters that are more affecting the model's output measure, epistemic uncertainty propagation may be applied to a simplified version of the model, i.e., the one that accounts only for the most important parameters.

# Part IV

# Other applications: IoT and Mobile CrowdSensing

CHAPTER 9

# Evaluation of Mobile CrowdSensing performance

## 9.1 Motivation

Internet of Things (IoT) is gaining more and more importance thanks to the recent advancements in mobile, pervasive sensing and communication technologies. It aims at connecting several physical objects into a unique and larger space, the *cyberspace* [64, 90], opening up new application scenarios that may involve billions of devices and users, a big challenge that must be properly addressed.

A promising attempt in this direction is Mobile CrowdSensing (MCS), which proposes to deploy and run IoT applications on mobiles by actively involving their owners in a volunteer/crowd-based fashion [56]. In fact, MCS allows to reach a large number of users, who may also act as contributors sharing their devices to mainly provide sensing facilities.

Several success stories confirm the effectiveness of the crowdsensing approach in IoT contexts [31, 46, 101], thus attracting interests from both academic and business communities that are investing resources and efforts to implement middleware mechanisms [115, 144] and to investigate on a commercial exploitation for MCS. This imposes to revise the MCS paradigm,

extending its scope to business contexts where SLAs on functional and non functional properties have to be enforced to meet quality of service (QoS) requirements. Therefore, proper mechanisms and tools for supporting the design and the operation of MCS services are required. In particular, modeling and evaluation techniques dealing with MCS contribution dynamics issues, such as churning (i.e., the random and unpredictable join and leave processes characterizing the contributors), must be developed.

The contributors' arrival and service processes, that identify the temporal behavior of an MCS system, have to be properly characterized recurring to stochastic models able to adequately represent their fluctuations.

The resulting model is often not memoryless and the corresponding stochastic process is non-Markovian, implying to take into account related memory issues in its analysis. This is particularly true in MCS systems where the service requests are first split into simple tasks, then assigned to contributor nodes whose processing could be interrupted and rescheduled due to churning. Sometimes, the results obtained by the partial processing of an interrupted task are not wasted, e.g., for purely sensing activities, or also in the case the MCS services implement checkpointing policies to limit the impact of rescheduling on QoS and performance. To take into account this aspect while modeling the system, i.e., to properly represent recovery from the conditions before the interruptions or from checkpoints, specific memory preservation mechanisms are required.

State space based solutions have been mainly proposed in literature to address these issues [32,42,68,69,95]. However, their main drawback is the well known *state space explosion*, when dealing with large-complex problems, limiting their applicability especially in MCS-IoT contexts, where the number of requests and contributors could easily reach thousands or even millions. Furthermore, none of them except [42] takes into account checkpointing policies and related issues, enabling the representation of contribution dynamics and churning, but not dealing with the memory problem.

## 9.2 A close-up view of MCS

MCS [56] is a computing paradigm where contributors share their mobile devices (e.g., smartphones, tablets, PDAs, laptops, etc.) with specific applications in order to gather and aggregate their data for further processing. From a high level perspective, the architecture of an MCS application mainly implements a client-server model as shown in Figure 9.1. The MCS application client deployed on each involved node (usually after it has been explicitly downloaded and installed by the contributor on his/her

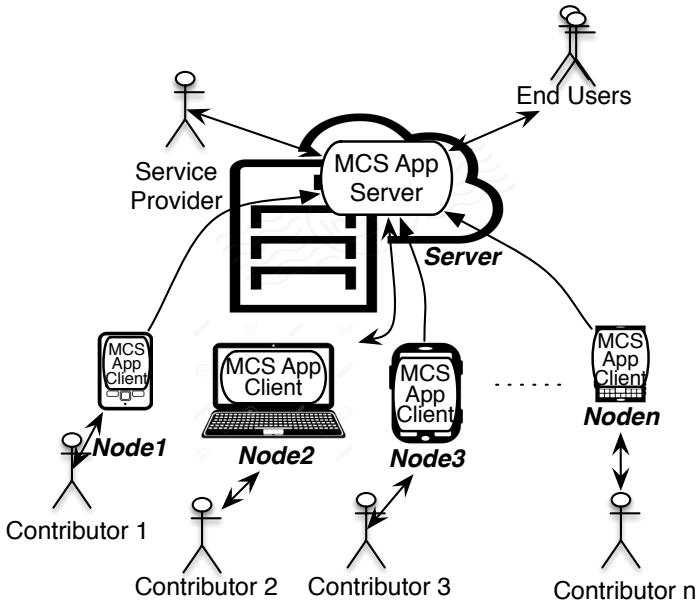



**Figure 9.1:** *MCS application scenario.*

device) produces, (pre-)processes and sends data to the server that collects, aggregates, further processes (if required) and stores such data, providing information and results to the end-users. More specifically, three different stakeholders can be identified:

- *contributors*, i.e., the owners of the devices that are shared to build up the MCS sensing infrastructure;
- the *service provider*, which manages the whole system and application by gathering, aggregating and processing data and results from contributing nodes;
- *end-users*, which use the MCS application to access the services and the information it provides.

These roles are not mutually exclusive and, for instance, contributors may also act as end-users and vice-versa.

In the MCS application processing, the main activities are decomposed into simpler *tasks* to be independently executed by the nodes. Each client may elaborate zero, one or many tasks, whose results are collected and recomposed by the application server. To prevent and minimize churning issues due to random and unpredictable joins and leaves of the contributors,

**Table 9.1:** *Metrics of interests for the three stakeholders of an MCS system.*

| ***PERSPECTIVE METRIC*** | ***Contributor*** | ***Service Provider*** | ***End User*** |
|---|---|---|---|
| ***Resource Utilization*** | Node | Server | - |
| ***Energy Consumption*** | Battery | Server | - |
| ***Response Time*** | - | Task | Service |
| ***Throughput*** | - | Task | |

an active contributing node may periodically send partially processed data (i.e., checkpoints) to restart future elaborations on other available nodes in the case of its departure. Indeed, the application client can sometimes predict the leave of a contributor (e.g., in the case of low battery).

The MCS paradigm has been already successfully adopted in several applications such as OpenStreetmap [65], as an effective solution for problems related to mobility [31, 46, 146], traffic monitoring [100, 101], public safety [3], smart cities [19] environment and pollution monitoring [146], emergency and crowd management [89], to mention but a few. Nevertheless, most of the potential of MCS is still untapped due to the limits of the contribution-based approach, often unreliable and unpredictable and therefore reflecting as high uncertainty in service fruition.

Thus, it is of strategic importance to focus on the overall MCS system rather than on a specific application, since the performance of the MCS application is strongly and mainly affected by the MCS system performance, reliability and energy related metrics, which are governed by the contribution dynamics.

Specifically, several metrics of interest for each stakeholder mentioned above may be identified and they are reported in Table 9.1. A contributor is mainly interested in knowing the impact of contribution on his/her device or node in terms of computing resource utilization (CPU, memory, storage) and battery charge. Instead, service provider is mainly interested in managing the system's resources, i.e., maximizing their utilization and reducing power consumption, while reducing the time for processing a task and increasing the system throughput. This also affects the time required for processing a request, which is the main metric of interest for the end-users.

A model or a modeling technique can provide a valid support to the

design, deployment and assessment of the MCS applications. However, several aspects must be taken into account to properly represent an MCS system. As said above, the most challenging one is the fluctuation of the number of contributors, which strongly impacts on the MCS underlying infrastructure reflecting on the non functional properties of the applications. Thus, an MCS system model has to primarily accounts for the contribution dynamics in the stochastic characterization of both arrival (join) and departure (leave) times of the contributors, by using specific probability distributions. Moreover, a stochastic characterization is required for the whole system, including the end-user's requests arrival process and service time. In general these are not Markovian, since their stochastic behavior is not memoryless. Furthermore, the model has also to deal with the complexity of an MCS system, for example, taking into account queuing effects as well as checkpointing policies.

## 9.3 Modeling MCS

The description given in Section 9.2 about the scenario considered in this chapter shows the need of a modeling technique that allows to evaluate the metrics of interest identified in Table 9.1, while dealing with churning issues. This strongly suggests to address the problem in a hierarchical and layered way, separating contribution aspects and concerns from provisioning and fruition ones.

For this purpose, considering the scenario in Figure 9.1, a two-layer modeling technique is proposed splitting the contribution and the processing subsystems. In the former the contribution dynamics due to node fluctuations joining and leaving the MCS system is mainly represented. Instead, the latter focuses on the processing of requests and tasks, therefore on the service dynamics.

However, the two subsystems are not independent since contributors mainly serve the requests and their tasks. Indeed, when a contributor joins the contribution network, a server in the processing subsystem is turned on and can serve the incoming requests. The server is turned off when the associated contributor leaves the contribution network. The connection between a server and a contributor lasts until the latter leaves the MCS system.

If there are not contributors associated to a server, then it may be turned on by the next incoming contributor. In fact, a contributor is connected to only a server during all its life, whereas several contributors may be associated to the same server during the whole simulation.

Another aspect to take into account in the MCS modeling is the sys-

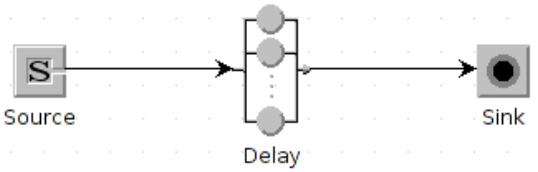



**Figure 9.2:** *The contribution queuing network sub-model.*

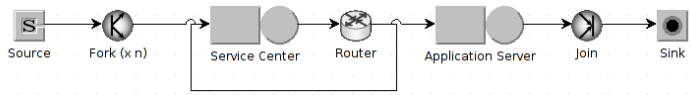



**Figure 9.3:** *The processing queuing network sub-model.*

tem complexity. Indeed, the number of users and contributors in an MCS system is commonly in the order of thousands, which excludes state space-based models. For this reason, QN formalism has been chosen due to its well known capability in dealing with complexity.

### 9.3.1 The contribution QN

The QN sub-model representing the MCS contribution dynamics is shown in Figure 9.2. The contributors can join and leave the MCS system randomly and the time they spend in the system is the contribution time. Therefore, they are stochastically represented by general distributions.

A job in the `Delay` station (i.e., a $G/G/\infty$ queue) represents a contributor node that is sharing its resources. The arrival rate to the `Delay` and the time spent by a job in it are characterized by general density. They represent the arrivals of contributors in the system and their contribution times (i.e., the time they are available to serve the end-users' requests), respectively. As discussed above, the `Delay` station of this network is connected to the `Service Center` resource in processing QN shown in Figure 9.3. In fact, the number of servers in the processing sub-model is equal to the number of jobs in the `Delay` station of the contribution QN.

### 9.3.2 The processing QN

The processing sub-model represents the incoming end-user requests that must be processed by the contributor nodes. Figure 9.3 shows the proposed QN sub-model for the processing part of the MCS system. As said, the overall QN model is composed of the two sub-models depicted in Figures 9.2 and 9.3 that are strictly interconnected.

The tasks arrival process is characterized by a general density, does not depend on the contribution QN and is modeled by the `Source` station. Also the `Application Server` station does not depend on the contribution QN and its service time is generally distributed, thus resulting in a $G/G/1$ queue. This resource models the commit of a request after all its tasks have been processed. Indeed, a request of an end-user is decomposed by the *Fork* into $n$ tasks that are served by the contributors in the MCS system. After all the tasks belonging to the same end-user's request have been processed, they go to the `Application Server` where they are further processed and finally joint (by the *Join* station) to provide the expected output.

The processing in the `Service Center` station depends on the contribution QN. To represent such a dependency, the `Service Center` is referred to as a $G/G/x$ queue with generally distributed arrival and service times and a variable number of servers $x \in [0, \infty)$, specifically representing the contribution churning dynamics. The number of servers $x$ is associated with the number of contributors that are in the contribution QN of Figure 9.2. Indeed, in order to represent the fluctuations of the number of servers in `Service Center`, a server is tuned on/off every time a contributor joins/leaves the contribution sub-model.

A task is served by only one server at a time but, since it could be rescheduled due to churning, it may be served by different servers until it is completed. Indeed, a task can exit the `Service Center` either after completion or if the $i$-th server that is processing that task is turned off, meaning that the associated contributor left the system. In the former case, the completed task is sent to the `Application Server` by the `Router` to be aggregated with the other $n - 1$ tasks of the same end-user's request. In the latter case, the task is rescheduled, thus sent back by the `Router` to the `Service Center`, waiting to be processed by an idle server.

Priority policies may be applied to the `Service Center` in order to prioritize dropped tasks. Since the number of available servers may be lower than the number of tasks that need to be served, only $x$ tasks can be served at the same time by the `Service Center`. This way, a task that has been interrupted can be rescheduled for processing with a higher priority. The rescheduling has also to take into account possible memory policies and strategies adopted to improve performance, such as the checkpointing one, thus implementing proper memory mechanisms to restart from checkpoints.

### 9.3.3 Output measures

Once the model has been specified, the performance and energy consumption metrics of interest may be identified in terms of QN measurements.

Referring to Table 9.1, an end-user is interested in the average *Push time*, i.e., the time required by the system to process an end-user request, that has been split into $n$ tasks. Therefore, this is the time for processing the $n$ tasks and their further elaboration in the `Application Server` after aggregation. Considering a generic end-user request $i$, its push time is derived as:

$$PushTime_i = max_{(j=1,..,n)}(R_i^j) \tag{9.1}$$

where $n$ is the number of tasks that the system collects before returning some results to the end-user and $R_i^j$ is the response time of the $j$-th task of the $i$-th request, obtained by analyzing the processing QN. Thus, the average *Push time* is:

$$PushTime = \frac{\sum_{i=1}^{I_{max}} PushTime_i}{I_{max}} \tag{9.2}$$

where $I_{max}$ is the number of requests processed by the system.

The service provider may be interested in the average system response time $R$ and throughput $X$. The former is obtained monitoring the time spent by each request into the system, evaluating how long it spends in the system, the latter as $X = C/T$ where $T$ is the observation time and $C$ is the number of requests that have been completed in that time.

Finally, a contributor is interested in knowing the impact of contribution on his/her devices, quantified by the utilization $U$ and the battery energy consumed by the MCS application during contribution. For this purpose, the average utilization $U$ of each node is evaluated as:

$$U = \frac{\sum_{i=0}^{x_{max}} ActivityTime_i}{\sum_{i=0}^{x_{max}} OnTime_i} \tag{9.3}$$

where $ActivityTime_i$ is the time spent by server $i$ processing a tasks, $OnTime_i$ is the total time that server $i$ was available for the MCS application (either busy serving a task or idle, waiting for a task to be served), and $x_{max}$ is the maximum number of server in the `Service Center`.

Assuming that each device is mainly used for computations, the average power consumption $P(U)$ of each node is evaluated through the linear power model described in Eq. (2.7). Once the power consumption has been computed, the average energy consumption $E$ is derived as:

$$E = P(U) \cdot S_C \tag{9.4}$$

where $S_C$ is the contributors' average service time, i.e., the average time each contributor spends into the system.

The average battery charge used by each contributor for sharing his/her resources with the MCS system is estimated as:

$$Battery\ consumption = \frac{E}{\hat{\gamma}} \tag{9.5}$$

where $\hat{\gamma}$ is the average battery capacity of the contributors' devices.

## 9.4 Validation and Evaluation

To explain in details the proposed technique, an MCS case study is analyzed in this section. Therefore, the parameters and configuration settings adopted for the analysis – they have been taken from existing literature when available – are discussed. In order to prove the validity of the QN model, it is compared with an analytic one. In particular, since a CTMC is used for comparison, the scope of the QN model has been restricted to the exponential case and the number of entities involved has been capped. These assumptions are relaxed for further investigations about the MCS system's performance and energy aspects.

### 9.4.1 Parameters and Configurations

As discussed in Section 9.3, each request submitted by end-users is split into $n$ simpler tasks that are assigned to and processed by contributor nodes. Once the $n$ tasks have been processed, the `Application Server` aggregates the results and sends the obtained outcomes to the end-users. To take into account checkpointing-restart policies, the MCS systems are investigated with and without memory strategies. In the former case, the checkpoint strategy lets the system recovers the work already done on each task when they are restarted, in the latter one the task processing restarts from scratch.

As stated above, the parameters used in the experiments have been taken from literature when possible. In particular, the arrival times of contributors and requests are exponentially distributed, as well as the service times of the `Delay` and `Application Server` stations. The service time of each server in the `Service Center` station is characterized by a 3-stage Erlang density, similarly to [42].

The average service time of the `Application Server`, $S_{AS}$, and `Delay`, $S_C$, are the same in all the experiments, i.e., 10 seconds and 30 minutes, respectively. The inter-arrival times of the two networks (i.e.,

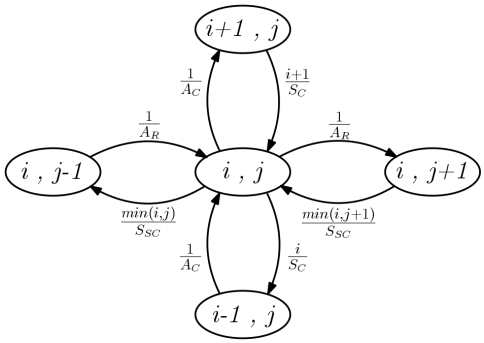



**Figure 9.4:** *Part of the CTMC modeling the MCS system used for validation.*

$A_C$ and $A_R$ for the contribution sub-model and the processing one, respectively) and the average service time of each server in the `Service Center`, $S_{SC}$, are changed for each experiment as follows: $A_C = \{1, 10, 20, 30\}$ minutes; $A_R = \{100, 200, 300\}$ minutes; $S_{SC} = \{5, 10, 15, 20, 25, 30, 35, 40\}$ minutes.

The strategy used in the `Service Center` to select the next task to be processed is FCFS with priority. The priority of a task is increased by the system every time the task is sent from the server back to the queue (i.e., it has not been completed by the current contributor). The task with the highest priority is the first to be served. If two or more tasks have identical priority, FCFS strategy is adopted. The number of tasks that must be committed before the system returns some results to end-user is $n = 10$ (i.e., the number of tasks generated by each request).

### 9.4.2 Model Validation

To validate the proposed model, we compare it against the analytic results obtained using the CTMC model, thus restricting the validation scope to the exponential case. Part of the CTMC model used for validation is depicted in Figure 9.4

Each state of this CTMC is characterized by a pair $(i, j)$ where $i$ and $j$ are the number of contributors and requests in the system, respectively. When the number of contributors changes, the system goes to state $(i-1, j)$ at rate $i/S_C$ if a contributor leaves the system or to state $(i + 1, j)$ at rate $1/A_C$ if a new one joins the MCS system. The state can move from $(i, j)$ to $(i, j - 1)$ with rate $min(i, j)/S_{SC}$ if a task is completed, or to $(i, j + 1)$ with rate $1/A_R$ if a new task enters the `Service Center`.

As said, in order to validate the QN model against the CTMC, the inter-

arrival and service times of contributors and requests are assumed to be exponentially distributed. This way, also the service time of each server in `Service Center` follows an exponential distributions and its average is $S_{SC}$.

The `Fork/Join` stations of the processing QN are neglected and the original $A_R$ is divided by the number $n$ of tasks generated by each request (i.e., $n = 10$).

In particular, only the case where $A_R = 20$ minutes (i.e., $200\ min./10$) and $A_C = 10$ minutes (i.e., 3 contributors into the system on average, since $S_C = 30$ minutes) has been evaluated.

The state space has been truncated assuming the number of contributors and requests cannot be larger than 20. Note that, that last assumption does not affect the results, since the system that is considered for validation rarely has a number of contributors and requests larger than 20.

Two different discrete event simulation tools have been used for the validation phase. OMNeT++ [137], an extensible, modular, component-based C++ simulation library and framework used for network simulations, and JMT [11]. All the performance indexes have been estimated with $99\%$ confidence intervals.

The results obtained by analyzing the simulative and analytic models are shown in Figure 9.5 that depicts the average contributors' utilization, the system response time and the average number of tasks in the `Service Center`. The Q-Q plots in Figures 9.5a, 9.5b and 9.5c show that the performance of the MCS system obtained by the CTMC and the QN model are almost the same. The *mean absolute percentage error* made by the QN model with respect to the CTMC one is plotted in Figure 9.5d, is computed as

$$\frac{|\Theta_{CTMC} - \Theta_{QN}|}{\Theta_{CTMC}} \qquad \Theta = \{U, R, N_{SC}\}$$

and is always lower than $4\%$.

In particular, the *average relative error* for utilization and system response time is lower than $1\%$, and it is lower than $1.26\%$ when considering the number of tasks in `Service Center`. Similar results are obtained when JMT is used to evaluate the QN model.

### 9.4.3 Evaluation

After validation, the QN model has been used to perform parametric experiments by varying either $A_C$ or $A_R$ and relaxing the exponential assumption. In particular, when $A_C$ ($A_R$) varies, $A_R$ is set to 200 minutes ($A_C$ is set to 20

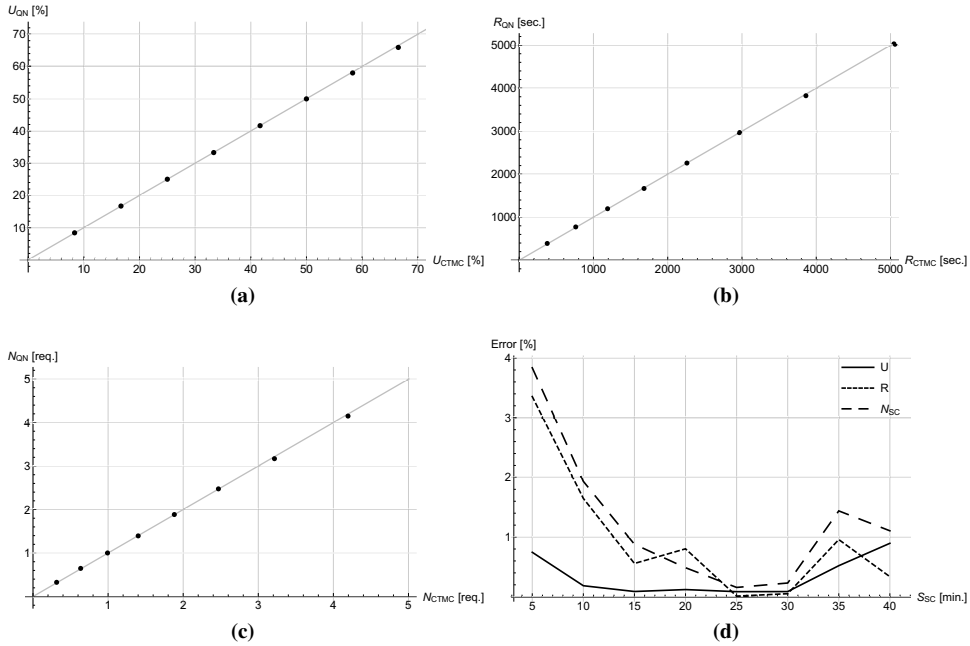



**Figure 9.5:** *Q-Q plots and relative error analysis to compare the performance indexes of the MCS system obtained with the QN model against the results obtained with the CTMC one. In particular, they depict (a) the average contributor's utilization, (b) system response time, (c) number of tasks in* `Service Center` *and (d) relative errors of QN model with respect to the CTMC one for the three metrics considered.*

minutes). In the experiments also $S_{SC}$ varies, assuming the tasks processed in *Service Center* may have different complexity and service demands. In fact, for each test performed and analyzed, two out of three parameters are varying.

With regard to the energy measurements, in the experiments we assume $P_{idle} = 0.268$ watt and $P_{max} = 1$ watt for each node, as estimated in [20] while analyzing the power consumption of smartphones. The battery capacity is assumed to be $\hat{\gamma} = 7.77$ watthour for all the devices.

In order to evaluate the QN model presented in Section 9.3, considering the non-exponential parameters and the performance indexes there specified, OMNeT++ has been extended implementing the dependency between the contribution and the processing sub-models and modifying the `Job` implementation to consider different memory strategies as discussed in Section 9.2. The MCS system here described is simulated for an observation time $T = 7$ days and the results obtained are discussed in the following, taking into account the different MCS stakeholders perspectives.

#### Contributor

Contributors are mainly interested in quantifying the impact of the contribution on their resources. After deriving the average utilization of their devices through Eq. (9.3), energy and battery consumption may be computed using Eqs. (9.4) and (9.5).

The results are shown in Figure 9.6, where the average utilization per contributor as a function of the average service time of `Service Center` is depicted.

In Figures 9.6a and 9.6b different contributors' inter-arrival times are considered, whereas the average arrival time of the requests is $A_R = 200$ minutes and the number of tasks generated by each new job is $n = 10$, Both checkpoint and no memory strategies are considered and compared in this analysis.

As expected, the nodes utilization decreases with the contribution inter-arrival time since the total number of contributors (i.e., $N_C = S_C/A_C$) increases when the inter-arrival time $A_C$ decreases, being the average contribution time constant, i.e., $S_C = 30$ minutes. Furthermore, if a checkpoint memory strategy is adopted, the utilization of each device is lower than the service without memory, since contributors have to work more in the latter case.

In Figures 9.6c and 9.6d the inter-arrival time of contributors is set to $20$ minutes, whereas the requests' arrival time varies. Also in this case, the contributors' utilization with checkpoint strategy is lower than the one

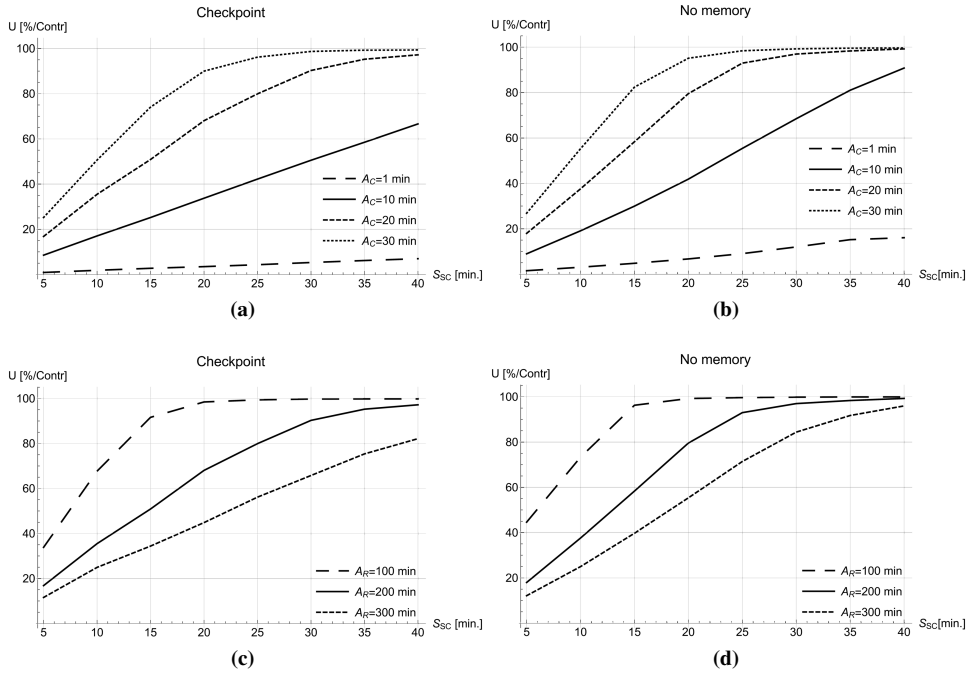



**Figure 9.6:** *Average utilization per contributor as a function of service time $S_{SC}$, with fixed average requests' inter-arrival time and either (a) checkpoints or (b) without memory, and fixed average contributors' inter-arrival time and either (c) checkpoints or (d) without memory.*

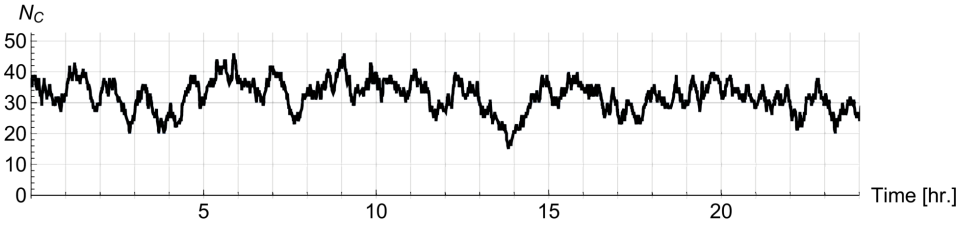



**Figure 9.7:** *Number of contributors in the MCS system in the case of $A_C = 1$ minute and $S_C = 30$ minutes, during a 24 hours observation window.*

with no memory. A higher utilization is also observed when the requests' average inter-arrival time is shorter since the number of requests into the system increases.

The utilization is then used to estimate the power consumption through the linear model described in Eq. (2.7), thus for computing the devices' energy consumption as described in Eq. (9.4). For this reason, energy consumption curves have the same trend of the utilization ones and they have not been plotted for the sake of order. However, using the parameters previously presented in this section, the battery depletion for a $30$ minutes contribution is never larger than $7\%$ of its capacity $\hat{\gamma}$, when $U = 100\%$.

#### Service provider

Several metrics of interest for the service provider can be obtained using the QN model. One of them is the number of contributors in the system, since the number of end-users' requests served by the MCS system depends on it. Figure 9.7 shows the contribution dynamic for $A_C = 1$ minute and $S_C = 30$ minutes. The evolution of the contribution QN is represented only for a day in stationary conditions. In this case, the number of contributors is in the range between $14$ and $47$, and its average is $30$.

Monitoring the system response time is important in order to check if the performance are degrading and the pledged QoS cannot be satisfied leading to SLAs violations. This metric is plotted in Figure 9.8 against the service time of the `Service Center`, still considering different memory strategies. In Figures 9.8a and 9.8b the average arrival time of the requests is set to $200$ minutes and the contributors' one varies. In the former figure, the checkpoint memory strategy is used, whereas in the latter one no memory strategies are adopted. As expected, the system response time stands in inverse proportion to the number of contributors into the system.

In fact, when enough contributors are sharing their resources, the tasks are served without interruption and they are completed in a shorter time.

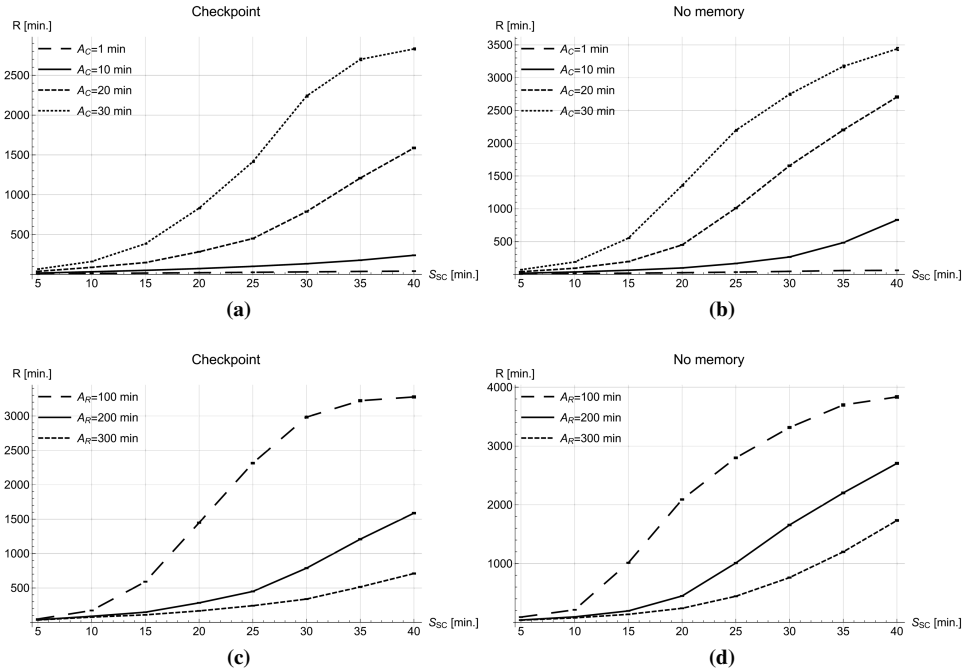



**Figure 9.8:** *Average system response time for tasks as a function of service time $S_{SC}$, with fixed average requests' inter-arrival time and either (a) checkpoints or (b) without memory, and fixed average contributors' inter-arrival time and either (c) checkpoints or (d) without memory.*

On the contrary, when the number of contributors is too small, each task may spend several time waiting to be served. This behavior is intensified without checkpointing, since a task must be processed from scratch every time it is rescheduled.

Figures 9.8c and 9.8d show the requests' system response time when the contributors' average arrival time is $20$ minutes and the requests' one varies. Similar considerations to the previous case can be drawn. In fact, the system response time is also in inverse proportion to the requests' arrival time.

To be sure not to incur any penalty, the provider is also interested in system throughput, since the provisioning is often constrained by SLAs on this performance metric. The system throughput is shown in Figure 9.9 as a function of the service time $S_{SC}$, where the checkpoint memory strategy is compared to the no memory one, also considering different inter-arrival time for contributors and requests. A larger number of contributors (i.e., shorter contributors' inter-arrival time) lets the system provide the largest throughput also when complex tasks (i.e., tasks that need a long time to be served) are processed. In particular, if the system is not saturating, the throughput of the requests is equal to the requests' arrival rate. The throughput has its maximum value when the tasks to be completed are simple, whereas it decreases when the tasks are more complex. Also in this case the checkpoint memory strategy positively impacts on the MCS system, ensuring larger throughput than MCS systems without memory.

#### End-user

End-users are mainly interested in push time, i.e., the time elapsed from the submission of a request to its completion. In the case-study considered in this section, the system has to process $n = 10$ tasks before returning the results to the users. The average push time obtained through the QN model analysis, as described in Eqs. (9.1) and (9.2), is shown in Figure 9.10 as a function of the service time $S_{SC}$, comparing the checkpoint and no memory strategies and considering different values for the average inter-arrival times of either the contributors or the requests.

In Figures 9.10a and 9.10b the contributors' inter-arrival time is varying. Assuming the average inter-arrival time of the requests is $200$ minutes, the push time behavior is similar to the response time one. The push time increases when the tasks are complex (i.e., for larger service time values) or when the number of available contributors is low (i.e., for larger values of $A_C$). It is worth noticing that the larger the number of requests in the system, the faster the push time increases. Indeed, tasks with the same

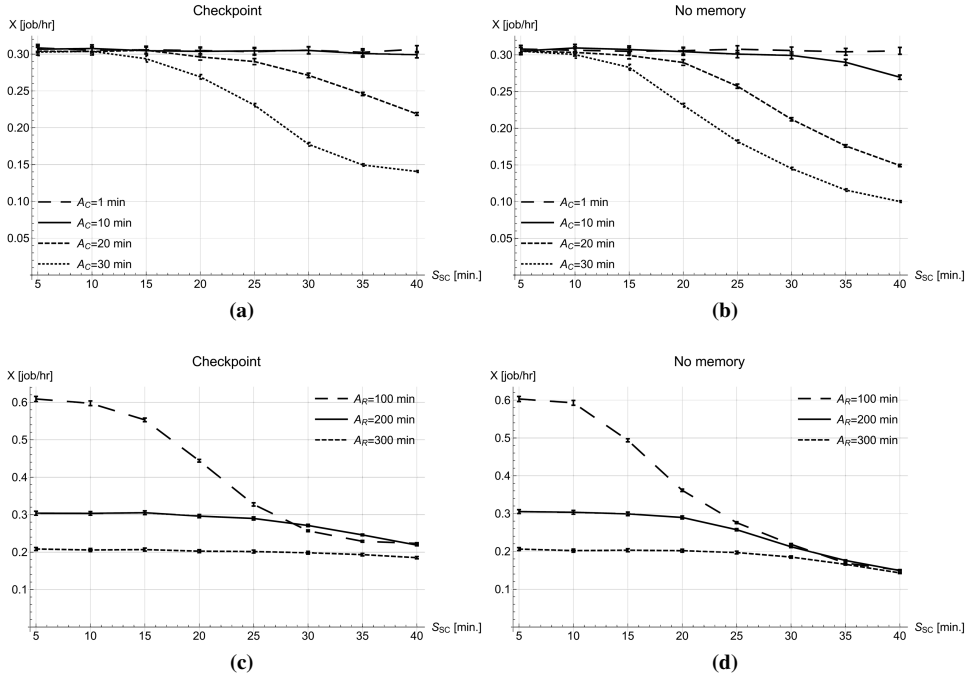



**Figure 9.9:** *Average system throughput as a function of service time $S_{SC}$, with fixed average requests' inter-arrival time and either (a) checkpoints or (b) without memory, and fixed average contributors' inter-arrival time and either (c) checkpoints or (d) without memory.*

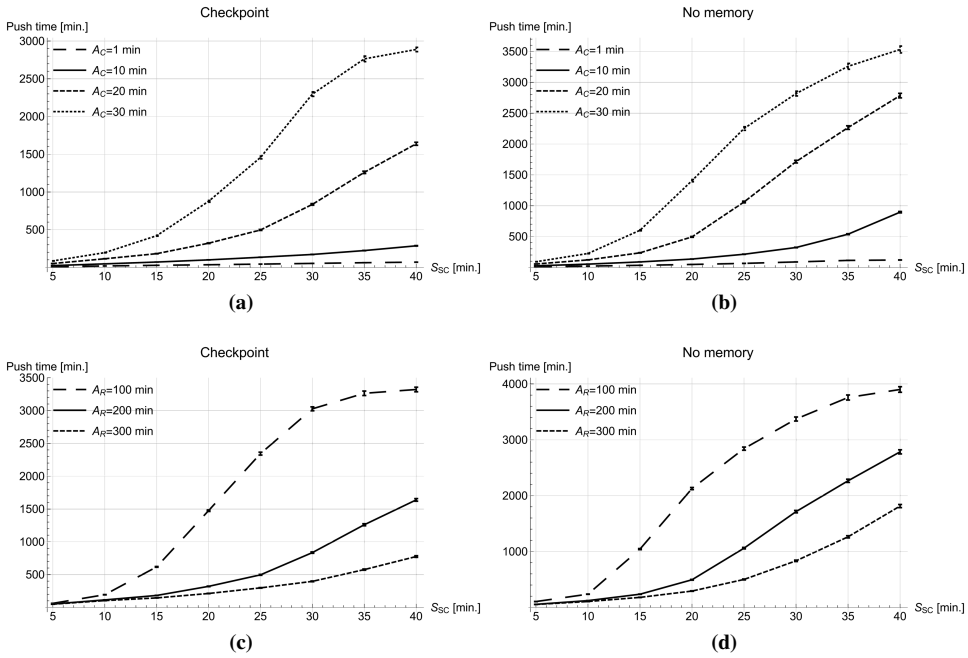



**Figure 9.10:** *Average push time as a function of service time $S_{SC}$, with fixed average requests' inter-arrival time and either (a) checkpoints or (b) without memory, and fixed average contributors' inter-arrival time and either (c) checkpoints or (d) without memory.*

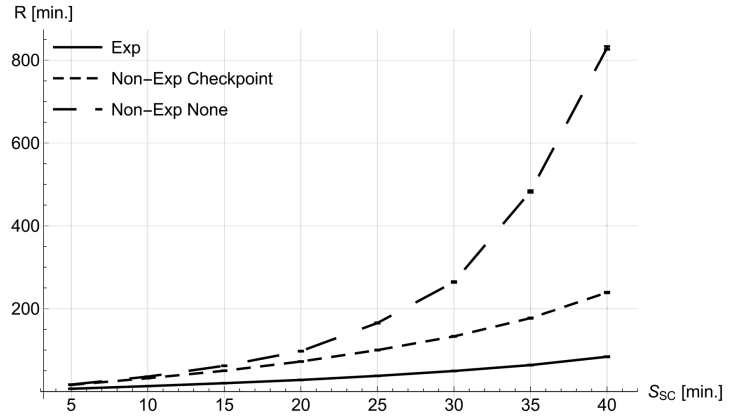



**Figure 9.11:** *Comparison of the system response time obtained with exponential and non-exponential models.*

complexity can easily affect the system performance when many other requests are already in the system. In both the considered cases, checkpoint strategy provides the best results.

#### Further results

The likelihood of exponential and non exponential models is here investigated by comparing the CTMC validation results to the results obtained with the non-Markovian QN model, where the service time of each server in `Service Center` is Erlang distributed and the `Fork/Join` stations are used to represent the end-users' requests decomposition.

For this purpose, we compare the system response times of the two models with $A_C = 10$ minutes and $A_R^{no-ex} = 200$ minutes, where the requests are split into $n = 10$ tasks ($A_R^{ex} = A_R^{no-ex}/n = 20$ minutes, for the exponential model). While studying the non-exponential model both the case with checkpoint memory strategy and the one without memory have been considered. They are compared to the exponential model that, being memoryless, cannot implement any memory policy.

Figure 9.11 depicts the system response time obtained with exponential and non-exponential models, showing that the QN model is required to get more accurate output measures.

CHAPTER 10

# Conclusion and future work

In this thesis, we have summarized our researches on data-center's energy consumption and new technologies for decreasing its impact on environment and organizations' costs. Different aspects have been considered, taking into consideration both modeling and estimation related problems.

A power model for CPU has been introduced; differently from the available ones, it takes into consideration how DVFS and SMT – two very widespread technologies used to improve the components' energy efficiency – affect the power consumption of the central processing unit. The results show that the proposed power model provides more accurate estimates than the already available ones that do no account for the two energy saving strategies.

A new metric, the *energy per time-unit of execution*, has been proposed to evaluate the energy efficiency of a system that executes a multi-class workload. In fact, many available metrics do not consider the different characteristics of the jobs while dealing with multi-class systems. The *energy per time-unit of execution* metric accounts for the heterogeneity of the jobs executed by the system, as well as the existing energetic relationship among all the components involved in the applications processing.

The *Pool depletion systems* – a new framework for studying perfor-

mance and energy consumption of Big Data applications – has been presented. It is used to analyze the applications that generates a large number of tasks that are then executed by parallel subsystems with a limited capacity. The most important metric for this kind of framework is *Depletion time*, i.e., the time required to complete the execution of all the tasks. Indeed, *Depletion time* minimization allows the application to be executed in a shorter time, thus service provider may decrease the energy consumption of the system and the users can reduce their fees.

While developing the *Pool depletion systems* framework, a new scheduling strategy has been investigated. It is the *Optimal population mix* strategy, that minimizes the *Depletion time* of an application making each subsystem work with its optimal population mix (i.e., the operating point for which the system provides its best performance). Such a strategy is interesting since it can decrease the *Depletion time* just modifying the execution sequence of the tasks. For this reason, the *Optimal population mix* strategy can be adopted together with already available policies to further decrease the execution time of Big Data applications. Some examples of such policies are the auto-scaling technique or the Completely Fair scheduler, that have been already used for optimizing the system utilization through optimal jobs allocation and minimization of the execution time of each single tasks. The results obtained through simulative analysis of *Pool depletion systems* have been validated with analytic models and experimental results on a private cloud infrastructure.

We have also taken into consideration epistemic uncertainty while using power models to study data-centers' energy efficiency. It is a kind of uncertainty due to a lack of knowledge and may be reduced observing a larger number of samples for each input parameter. In this thesis, epistemic uncertainty has been also considered with regard to M/M/1 queue – a model often adopted to represent servers and the data-center's components – in order to evaluate how uncertainty on the input parameters (i.e., inter-arrival and service rates) affects its average response time and number of customers. Moreover, an interesting relationship between epistemic uncertainty and parametric sensitivity has been discovered. Since the latter analysis requires shorter time and fewer computational power than the former one, it may help system designers in taking decisions about their systems.

Besides data-centers, also small devices' performance and energy consumption have been briefly taken into account. In fact, Internet of Things and sensors are one of the main reasons energy consumption has gained such importance in the last few years. In particular, a queuing network

model has been proposed to investigate the performance of Mobile Crowd-Sensing, a paradigm that allows to reach a large number of users that may also act as contributors by sharing their devices.

Since several aspects have been considered in this thesis, also future work may follow different directions in order to improve the work here presented.

*Pool depletion systems* may be extended accounting for multiple (i.e., more than two) classes of tasks and subsystems' resources. This way, further optimizations and limits of the system considered may be discovered and exploited. Moreover, it is also interesting to investigate how multi-core resources (e.g., multi-core CPU) are affecting the depletion time of this type of applications.

The *Optimal population mix* strategy should be integrated into some existing frameworks (e.g., Apache Spark, MapReduce) in order to evaluate the gains provided by that strategy in a real cloud environment. Since *Optimal population mix* strategy is strictly related to *Pool depletion systems*, the scheduling strategy should be also improved taking into consideration the results obtained by extending the *Pool depletion systems* framework as previously described.

Finally, the results on epistemic uncertainty may be extended taking into account different epistemic probability density functions for the input parameters. Indeed, in this thesis only Uniform and Erlang distributed input random variables have been considered. Moreover, also the relationship between epistemic uncertainty and parametric sensitivity should be further investigated considering different system configurations and input parameters' distribution.

# Bibliography


[1] Ibrahim Alagöz, Christoffer Löffler, Vitali Schneider, and Reinhard German. Simulating the energy management on smartphones using hybrid modeling techniques. In *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*, pages 207–224. Springer, 2014.

[2] Susanne Albers and Hiroshi Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms (TALG)*, 3(4):49, 2007.

[3] Elian Aubry, Thomas Silverston, Abdelkader Lahmadi, and Olivier Festor. Crowdout: a mobile crowdsourcing service for road safety in digital cities. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on*, pages 86–91. IEEE, 2014.

[4] Ha R. Bae, Ramana V. Grandhi, and Robert A. Canfield. Epistemic uncertainty quantification techniques including evidence theory for large-scale structures. *Computers & Structures*, 82(13–14):1101–1112, 2004.

[5] Gianfranco Balbo and Giuseppe Serazzi. Asymptotic analysis of multiclass closed queueing networks: Multiple bottlenecks. *Performance Evaluation*, 30(3):115–152, 1997.

[6] Enrico Barbierato, Marco Gribaudo, and Daniele Manini. Fluid approximation of pool depletion systems. In *International Conference on Analytical and Stochastic Modeling Techniques and Applications*, pages 60–75. Springer, 2016.

[7] Philipp Baumgärtel, Gregor Endler, Andreas M Wahl, and Richard Lenz. Inverse uncertainty propagation for demand driven data acquisition. In *Proceedings of the 2014 Winter Simulation Conference*, pages 710–721. IEEE, 2014.

[8] Christian Belady, Andy Rawson, John Pfleuger, and Tahir Cader. Green grid data center power efficiency metrics: Pue and dcie. Technical report, Technical report, Green Grid, 2008.

[9] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future generation computer systems*, 28(5):755–768, 2012.

[10] Anton Beloglazov, Rajkumar Buyya, Young Choon Lee, Albert Zomaya, et al. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in computers*, 82(2):47–111, 2011.

[11] Marco Bertoli, Giuliano Casale, and Giuseppe Serazzi. Jmt: performance engineering tools for system modeling. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):10–15, 2009.

[12] Mark Blackburn and Green Grid. Five ways to reduce data center server power consumption. *The Green Grid*, 42:12, 2008.

[13] Stephen M Blackburn, Robin Garner, Chris Hoffmann, Asjad M Khang, Kathryn S McKinley, Rotem Bentzur, Amer Diwan, Daniel Feinberg, Daniel Frampton, Samuel Z Guyer, et al. The dacapo benchmarks: Java benchmarking development and analysis. In *ACM Sigplan Notices*, volume 41, pages 169–190. ACM, 2006.

[14] Ata E Husain Bohra and Vipin Chaudhary. Vmeter: Power modelling for virtualized clouds. In *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8. Ieee, 2010.

[15] Luca Bortolussi and Nicolas Gast. Mean field approximation of uncertain stochastic models. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 287–298. IEEE, 2016.

[16] Dejene Boru, Dzmitry Kliazovich, Fabrizio Granelli, Pascal Bouvry, and Albert Y Zomaya. Energy-efficient data replication in cloud computing datacenters. *Cluster computing*, 18(1):385–402, 2015.

[17] Onno Johan Boxma and IA Kurkova. The m/m/1 queue in a heavy-tailed random environment. *Statistica Neerlandica*, 54(2):221–236, 2000.

[18] Rajkumar Buyya, Anton Beloglazov, and Jemal Abawajy. Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges. *arXiv preprint arXiv:1006.0308*, 2010.

[19] Giuseppe Cardone, Luca Foschini, Paolo Bellavista, Antonio Corradi, Cristian Borcea, Manoop Talasila, and Reza Curtmola. Fostering participaction in smart cities: a geo-social crowdsensing platform. *IEEE Communications Magazine*, 51(6):112–119, 2013.

[20] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *USENIX annual technical conference*, volume 14. Boston, MA, 2010.

[21] Emiliano Casalicchio, Lars Lundberg, and Sogand Shirinbab. Energy-aware adaptation in managed cassandra datacenters. In *Cloud and Autonomic Computing (ICCAC), 2016 International Conference on*, pages 60–71. IEEE, 2016.

[22] Davide Cerotti, Marco Gribaudo, Pietro Piazzolla, Riccardo Pinciroli, and Giuseppe Serazzi. Multi-class queuing networks models for energy optimization. In *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools*, pages 98–105. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014.

[23] Davide Cerotti, Marco Gribaudo, Pietro Piazzolla, Riccardo Pinciroli, and Giuseppe Serazzi. Modeling power consumption in multicore cpus with multithreading and frequency scaling. In *Information Sciences and Systems 2015*, pages 81–90. Springer, Cham, 2016.

[24] Davide Cerotti, Marco Gribaudo, Riccardo Pinciroli, and Giuseppe Serazzi. Stochastic analysis of energy consumption in pool depletion systems. In *International GI/ITG Conference on Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*, pages 25–39. Springer, Cham, 2016.

[25] Davide Cerotti, Marco Gribaudo, Riccardo Pinciroli, and Giuseppe Serazzi. Optimal population mix in pool depletion systems with two-class workload. In *10th EAI International Conference on Performance Evaluation Methodologies and Tools*, pages 11–18. ACM, 2017.

[26] Venkatesan T Chakaravarthy, Anamitra Roy Choudhury, Sambuddha Roy, and Yogish Sabharwal. Scheduling jobs with multiple non-uniform tasks. In *European Conference on Parallel Processing*, pages 90–101. Springer, 2013.

[27] Hung-Ching Chang, Bo Li, Matthew Grove, and Kirk W Cameron. How processor speedups can slow down i/o performance. In *Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2014 IEEE 22nd International Symposium on*, pages 395–404. IEEE, 2014.

[28] Jeffrey S Chase, Darrell C Anderson, Prachi N Thakar, Amin M Vahdat, and Ronald P Doyle. Managing energy and server resources in hosting centers. *ACM SIGOPS operating systems review*, 35(5):103–116, 2001.

[29] Yiyu Chen, Amitayu Das, Wubi Qin, Anand Sivasubramaniam, Qian Wang, and Natarajan Gautam. Managing server energy and operational costs in hosting centers. In *ACM SIGMETRICS performance evaluation review*, volume 33, pages 303–314. ACM, 2005.

[30] Kihwan Choi, Ramakrishna Soma, and Massoud Pedram. Dynamic voltage and frequency scaling based on workload decomposition. In *Proceedings of the 2004 international symposium on Low power electronics and design*, pages 174–179. ACM, 2004.

[31] Yohan Chon, Nicholas D Lane, Fan Li, Hojung Cha, and Feng Zhao. Automatically characterizing places with opportunistic crowdsensing using smartphones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 481–490. ACM, 2012.

[32] Marco Conti, Chiara Boldrini, Salil S Kanhere, Enzo Mingozzi, Elena Pagani, Pedro M Ruiz, and Mohamed Younis. From manet to people-centric networking: milestones and open research challenges. *Computer Communications*, 71:1–21, 2015.

[33] Giovanni Cosulich, Pierluigi Firpo, and Stefano Savio. Power electronics reliability impact on service dependability for railway systems: a real case study. In *Industrial Electronics, 1996. ISIE'96., Proceedings of the IEEE International Symposium on*, volume 2, pages 996–1001. IEEE, 1996.

[34] Matthew Daigle and Chetan S Kulkarni. End-of-discharge and end-of-life prediction in lithium-ion batteries with electrochemistry-based aging models. In *AIAA Infotech@ Aerospace*, page 2132. 2016.

[35] Howard David, Chris Fallin, Eugene Gorbatov, Ulf R Hanebutte, and Onur Mutlu. Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the 8th ACM international conference on Autonomic computing*, pages 31–40. ACM, 2011.

[36] Miyuru Dayarathna, Yonggang Wen, and Rui Fan. Data center energy consumption modeling: A survey. *IEEE Communications Surveys & Tutorials*, 18(1):732–794, 2016.

[37] G Dazi, Stefano Savio, and Pierluigi Firpo. Estimate of components reliability and maintenance strategies impact on trains delay. In *Proc 21st European Conf on Modelling and Simulation ECMS*, 2007.

[38] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[39] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[40] Peter J Denning and Jeffrey P Buzen. The operational analysis of queueing network models. *ACM Computing Surveys (CSUR)*, 10(3):225–261, 1978.

[41] Daniel Díaz-Sánchez, Andrés Marín-López, Florina Almenarez, Rosa Sánchez-Guerrero, and Patricia Arias. A distributed transcoding system for mobile video delivery. In *Wireless and Mobile Networking Conference (WMNC), 2012 5th Joint IFIP*, pages 10–16. IEEE, 2012.

[42] Salvatore Distefano, Francesco Longo, and Marco Scarpa. Qos assessment of mobile crowd-sensing services. *Journal of Grid computing*, 14, 2016.

[43] John C Doyle, Joseph E Wall, and Gunter Stein. Performance and robustness analysis for structured uncertainty. In *Decision and Control, 1982 21st IEEE Conference on*, volume 21, pages 629–636. IEEE, 1982.

[44] Marc Doyle, Thomas F Fuller, and John Newman. Modeling of galvanostatic charge and discharge of the lithium/polymer/insertion cell. *Journal of the Electrochemical Society*, 140(6):1526–1533, 1993.

[45] Dimitris Economou, Suzanne Rivoire, Christos Kozyrakis, and Partha Ranganathan. Full-system power analysis and modeling for server environments. International Symposium on Computer Architecture-IEEE, 2006.

[46] Shane B Eisenman, Emiliano Miluzzo, Nicholas D Lane, Ronald A Peterson, Gahng-Seop Ahn, and Andrew T Campbell. Bikenet: A mobile sensing system for cyclist experience mapping. *ACM Transactions on Sensor Networks (TOSN)*, 6(1):6, 2009.

[47] 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. *ACM SIGMETRICS Performance Evaluation Review*, 40(1):163–174, 2012.

[48] Elmootazbellah N Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy-efficient server clusters. In *PACS*, volume 2325, pages 179–196. Springer, 2002.

[49] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *ACM SIGARCH Computer Architecture News*, volume 35(2), pages 13–23. ACM, 2007.

[50] Qiu Fang, Jun Wang, and Qi Gong. Qos-driven power management of data centers via model predictive control. *IEEE Transactions on Automation Science and Engineering*, 13(4):1557–1566, 2016.

[51] Ricardo M Fricks and Kishor S Trivedi. Importance analysis with markov chains. In *Reliability and Maintainability Symposium, 2003. Annual*, pages 89–95. IEEE, 2003.

[52] Alessandro Frossi. Policloud, user manual. Available at `http://policloud.polimi.it/wp-content/uploads/2015/04/PolicloudUserManual.pdf`, April 2015. Accessed: 2017-08-28.

[53] Anshul Gandhi, Varun Gupta, Mor Harchol-Balter, and Michael A Kozuch. Optimality analysis of energy-performance trade-off for server farm management. *Performance Evaluation*, 67(11):1155–1171, 2010.

[54] Anshul Gandhi, Mor Harchol-Balter, Rajarshi Das, and Charles Lefurgy. Optimal power allocation in server farms. In *ACM SIGMETRICS Performance Evaluation Review*, volume 37, pages 157–168. ACM, 2009.

[55] M Ganeshalingam, A Shehabi, and LB Desroches. Shining a light on small data centers in the us. 2017.

[56] Raghu K Ganti, Fan Ye, and Hui Lei. Mobile crowdsensing: current state and future challenges. *Communications Magazine, IEEE*, 49(11):32–39, 2011.

[57] Saurabh Kumar Garg, Chee Shin Yeo, Arun Anandasivam, and Rajkumar Buyya. Environment-conscious scheduling of hpc applications on distributed cloud-oriented data centers. *Journal of Parallel and Distributed Computing*, 71(6):732–749, 2011.

[58] Erol Gelenbe and Catherine Rosenberg. Queues with slowly varying arrival and service processes. *Management Science*, 36(8):928–937, 1990.

[59] Rahul Ghosh, Francesco Longo, Ruofan Xia, Vijay K Naik, and Kishor S Trivedi. Stochastic model driven capacity planning for an infrastructure-as-a-service cloud. *IEEE Transactions on Services Computing*, 7(4):667–680, 2014.

[60] Ricardo Gonzalez and Mark Horowitz. Energy dissipation in general purpose microprocessors. *Solid-State Circuits, IEEE Journal of*, 31(9):1277–1284, 1996.

[61] Steve Greenberg, Amit Khanna, and William Tschudi. High performance computing with high efficiency. *ASHRAE Transactions*, 115(2), 2009.

[62] Marco Gribaudo. *Theory and Application of Multi-formalism Modeling*. IGI Global, 2013.

[63] Marco Gribaudo, Riccardo Pinciroli, and Kishor Trivedi. Epistemic uncertainty propagation in power models. *Electronic Notes in Theoretical Computer Science*, 2017.

[64] Bin Guo, Daqing Zhang, Zhu Wang, Zhiwen Yu, and Xingshe Zhou. Opportunistic iot: Exploring the harmonious interaction between human and the internet of things. *Journal of Network and Computer Applications*, 36(6):1531–1539, 2013.

[65] Mordechai Haklay and Patrick Weber. Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, 7(4):12–18, 2008.

[66] DM Hamby. A review of techniques for parameter sensitivity analysis of environmental models. *Environmental monitoring and assessment*, 32(2):135–154, 1994.

[67] Abdul Hameed, Alireza Khoshkbarforoushha, Rajiv Ranjan, Prem Prakash Jayaraman, Joanna Kolodziej, Pavan Balaji, Sherali Zeadally, Qutaibah Marwan Malluhi, Nikos Tziritas, Abhinav Vishnu, et al. A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems. *Computing*, 98(7):751–774, 2016.

[68] Yang Han and Yanmin Zhu. Profit-maximizing stochastic control for mobile crowd sensing platforms. In *2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems*, pages 145–153. IEEE, 2014.

[69] Yang Han, Yanmin Zhu, and Jiadi Yu. Utility-maximizing data collection in crowd sensing: An optimal scheduling approach. In *Sensing, Communication, and Networking (SECON), 2015 12th Annual IEEE International Conference on*, pages 345–353. IEEE, 2015.

[70] Sunderesh S Heragu. *Facilities design*, chapter A.3.3, pages 539–541. CRC Press, 2008.

[71] Holger Hermanns, Jan Krčál, and Gilles Nies. How is your satellite doing? battery kinetics with recharging and uncertainty. *Leibniz Transactions on Embedded Systems*, 4(1):04–1, 2017.

[72] Lan Huang, Xiao-wei Wang, Yan-dong Zhai, and Bin Yang. Extraction of user profile based on the hadoop framework. In *Wireless Communications, Networking and Mobile Computing, 2009. WiCom'09. 5th International Conference on*, pages 1–6. IEEE, 2009.

[73] Jannik Hüels and Anne Remke. Energy storage in smart homes: Grid-convenience versus self-use and survivability. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2016 IEEE 24th International Symposium on*, pages 385–390. IEEE, 2016.

[74] Wolfram Research, Inc. Mathematica, Version 11.1. Champaign, IL, 2017.

[75] James R Jackson. Networks of waiting lines. *Operations research*, 5(4):518–521, 1957.

[76] Xibo Jin, Fa Zhang, Athanasios V Vasilakos, and Zhiyong Liu. Green data centers: A survey, perspectives, and future directions. *arXiv preprint arXiv:1608.00687*, 2016.

[77] Marijn R Jongerden and Boudewijn R Haverkort. Which battery model to use? *IET software*, 3(6):445–457, 2009.

[78] Frank P Kelly. Networks of queues with customers of different types. *Journal of Applied Probability*, 12(3):542–554, 1975.

[79] Atefeh Khosravi, Lachlan Andrew, and Rajkumar Buyya. Dynamic vm placement method for minimizing energy and carbon cost in geographically distributed cloud data centers. *IEEE Transactions on Sustainable Computing*, 2017.

[80] Leonard Kleinrock. Power and deterministic rules of thumb for probabilistic problems in computer communications. In *Proceedings of the International Conference on Communications*, volume 43, pages 1–43, 1979.

[81] Alexey Kopytov. Sysbench. `https://github.com/akopytov/sysbench`.

[82] Amogh Pramod Kulkarni and Mahesh Khandewal. Survey on hadoop and introduction to yarn. *International Journal of Emerging Technology and Advanced Engineering*, 4(5):82–87, 2014.

[83] Dara Kusic, Jeffrey O Kephart, James E Hanson, Nagarajan Kandasamy, and Guofei Jiang. Power and performance management of virtualized computing environments via lookahead control. *Cluster computing*, 12(1):1–15, 2009.

[84] ED Laowska, John Zahorjan, G Scott Graham, and Kenneth C Sevcik. Quantitative system performance, 1984.

[85] Kien Le, Ozlem Bilgir, Ricardo Bianchini, Margaret Martonosi, and Thu D Nguyen. Managing the cost, energy consumption, and carbon footprint of internet services. In *ACM SIGMETRICS Performance Evaluation Review*, volume 38, pages 357–358. ACM, 2010.

[86] Etienne Le Sueur and Gernot Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. 2010.

[87] Yingmin Li, David Brooks, Zhigang Hu, Kevin Skadron, and Pradip Bose. Understanding the energy efficiency of simultaneous multithreading. In *Proceedings of the 2004 international symposium on Low power electronics and design*, pages 44–49. ACM, 2004.

[88] Stavros Lopatatzidis, Jasper De Bock, Gert De Cooman, Stijn De Vuyst, and Joris Walraevens. Robust queueing theory: an initial study using imprecise probabilities. *Queueing Systems*, 82(1-2):75, 2016.

[89] Thomas Ludwig, Christian Reuter, Tim Siebigteroth, and Volkmar Pipek. Crowdmonitor: Mobile crowd sensing for assessing physical and digital activities of citizens during emergencies. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 4083–4092. ACM, 2015.

[90] Hua-Dong Ma. Internet of things: Objectives and scientific challenges. *Journal of Computer science and Technology*, 26(6):919–924, 2011.

[91] Rubens Matos, Jean Araujo, Danilo Oliveira, Paulo Maciel, and Kishor Trivedi. Sensitivity analysis of a hierarchical model of mobile cloud computing. *Simulation Modelling Practice and Theory*, 50:151–164, 2015.

[92] Indika Meedeniya, Irene Moser, Aldeida Aleti, and Lars Grunske. Architecture-based reliability evaluation under uncertainty. In *Proceedings of the joint ACM SIGSOFT conference–QoSA and ACM SIGSOFT symposium–ISARCS on Quality of software architectures–QoSA and architecting critical systems–ISARCS*, pages 85–94. ACM, 2011.

[93] Gerhard Ingmar Meijer. Cooling energy-hungry data centers. *Science*, 328(5976):318–319, 2010.

[94] David Meisner, Junjie Wu, and Thomas F Wenisch. Bighouse: A simulation infrastructure for data center systems. In *Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium on*, pages 35–45. IEEE, 2012.

[95] Giovanni Merlino, Stamatis Arkoulis, Salvatore Distefano, Chrysa Papagianni, Antonio Puliafito, and Symeon Papavassiliou. Mobile crowdsensing as a service: a platform for applications on top of sensing clouds. *Future Generation Computer Systems*, 56:623–639, 2016.

[96] Mark P Mills. The cloud begins with coal. *Digital Power Group. Online at: http://www. tech-pundit. com/wp-content/uploads/2013/07/Cloud_Begins_With_Coal. pdf*, 2013.

[97] Kesari Mishra, Kishor Trivedi, and Raphael Some. Uncertainty analysis of the remote exploration and experimentation system. *J. of Spacecraft and Rockets*, 49(6):1032–1042, 2012.

[98] Kesari Mishra and Kishor S Trivedi. Uncertainty propagation through software dependability models. In *Software Reliability Engineering (ISSRE), 2011 IEEE 22nd International Symposium on*, pages 80–89. IEEE, 2011.

[99] Kesari Mishra and Kishor S Trivedi. Closed-form approach for epistemic uncertainty propagation in analytic models. In *Stochastic Reliability and Maintenance Modeling*, pages 315–332. Springer, 2013.

[100] Waze Mobile. Waze website. `https://www.waze.com/`. Accessed: 2017-08-21.

[101] Prashanth Mohan, Venkata N Padmanabhan, and Ramachandran Ramjee. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 323–336. ACM, 2008.

[102] Douglas C Montgomery and George C Runger. *Applied statistics and probability for engineers*. John Wiley & Sons, 2010.

[103] Jogesh K Muppala, Manish Malhotra, and Kishor S Trivedi. Markov dependability models of complex systems: Analysis techniques. In *Reliability and Maintenance of Complex Systems*, pages 442–486. Springer, 1996.

[104] Xiaoming Nan, Yifeng He, and Ling Guan. Optimal resource allocation for multimedia cloud based on queuing model. In *Multimedia signal processing (MMSP), 2011 IEEE 13th international workshop on*, pages 1–6. IEEE, 2011.

[105] Ripal Nathuji and Karsten Schwan. Virtualpower: coordinated power management in virtualized enterprise systems. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 265–278. ACM, 2007.

[106] Anne-Cécile Orgerie. Green computing and sustainability. In *Energie et radiosciences-Journées scientifiques URSI France*, 2016.

[107] Sangyoung Park, Jaehyun Park, Donghwa Shin, Yanzhi Wang, Qing Xie, Massoud Pedram, and Naehyuck Chang. Accurate modeling of the delay and energy overhead of dynamic voltage and frequency scaling in modern microprocessors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(5):695–708, 2013.

[108] M Elisabeth Paté-Cornell. Uncertainties in risk analysis: Six levels of treatment. *Reliability Engineering & System Safety*, 54(2-3):95–111, 1996.

[109] Christy Pettey. Gartner estimates ict industry accounts for 2 percent of global co2 emissions. *Dostupno na: https://www. gartner. com/newsroom/id/503867*, 14:2013, 2007.

[110] Riccardo Pinciroli and Salvatore Distefano. Characterization and evaluation of mobile crowdsensing performance and energy indicators. *ACM SIGMETRICS Performance Evaluation Review*, 44(4):80–90, 2017.

[111] Riccardo Pinciroli and Salvatore Distefano. Extending queuing networks to assess mobile crowdsensing application performance. In *10th EAI International Conference on Performance Evaluation Methodologies and Tools*, pages 255–262. ACM, 2017.

[112] Riccardo Pinciroli, Marco Gribaudo, and Giuseppe Serazzi. Modeling multiclass task-based applications on heterogeneous distributed environments. In *International Conference on Analytical and Stochastic Modeling Techniques and Applications*, pages 166–180. Springer, Cham, 2017.

[113] Riccardo Pinciroli, Kishor Trivedi, and Andrea Bobbio. Parametric sensitivity and uncertainty propagation in dependability models. In *10th EAI International Conference on Performance Evaluation Methodologies and Tools*, pages 44–51. ACM, 2017.

[114] Eduardo Pinheiro, Ricardo Bianchini, Enrique V Carrera, and Taliver Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *Workshop on compilers and operating systems for low power*, volume 180, pages 182–195. Barcelona, Spain, 2001.

[115] Moo-Ryong Ra, Bin Liu, Tom F La Porta, and Ramesh Govindan. Medusa: A programming framework for crowd-sensing applications. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 337–350. ACM, 2012.

[116] Chuangang Ren, Di Wang, Bhuvan Urgaonkar, and Anand Sivasubramaniam. Carbon-aware energy capacity planning for datacenters. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on*, pages 391–400. IEEE, 2012.

[117] Catherine Rosenberg, Ravi Mazumdar, and Leonard Kleinrock. On the analysis of exponential queuing systems with randomly changing arrival rates: stability conditions and finite buffer scheme with a resume level. *Performance Evaluation*, 11(4):283–292, 1990.

[118] Emilia Rosti, Francesco Schiavoni, and Giuseppe Serazzi. Queueing network models with two classes of customers. In *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 1997. MASCOTS'97., Proceedings Fifth International Symposium on*, pages 229–234. IEEE, 1997.

[119] Mohamed Sallak, Sébastien Destercke, Walter Schön, Frédéric Vanderhaegen, Denis Berdjag, and Christophe Simon. Uncertainty, elicitation of experts' opinion, and human failures: Challenges for ram analysis of ertms sos. In *System of Systems Engineering Conference (SoSE), 2015 10th*, pages 88–93. IEEE, 2015.

[120] Andrea Saltelli, Marco Ratto, Terry Andres, Francesca Campolongo, Jessica Cariboni, Debora Gatelli, Michaela Saisana, and Stefano Tarantola. *Global sensitivity analysis: the primer*. John Wiley & Sons, 2008.

[121] Dimitri Scheftelowitsch, Peter Buchholz, Vahid Hashemi, and Holger Hermanns. Multi-objective approaches to markov decision processes with uncertain transition parameters. *arXiv preprint arXiv:1710.08986*, 2017.

[122] Koushik Sen, Mahesh Viswanathan, and Gul Agha. Model-checking markov chains in the presence of uncertainties. In *TACAS*, volume 3920, pages 394–410. Springer, 2006.

[123] John S Seng, Dean M Tullsen, and George ZN Cai. Power-sensitive multithreaded architecture. In *Computer Design, 2000. Proceedings. 2000 International Conference on*, pages 199–206. IEEE, 2000.

[124] Arman Shehabi, Sarah Smith, Dale Sartor, Richard Brown, Magnus Herrlin, Jonathan Koomey, Eric Masanet, Nathaniel Horner, Inês Azevedo, and William Lintner. United states data center energy usage report. 2016.

[125] Nozer D Singpurwalla. *Reliability and risk: a Bayesian perspective*. John Wiley & Sons, 2006.

[126] Damjan Škulj. Discrete time markov chains with interval probabilities. *International journal of approximate reasoning*, 50(8):1314–1329, 2009.

[127] Ying Song, Hui Wang, Yaqiong Li, Binquan Feng, and Yuzhong Sun. Multi-tiered on-demand resource scheduling for vm-based data center. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 148–155. IEEE Computer Society, 2009.

[128] Shekhar Srikantaiah, Aman Kansal, and Feng Zhao. Energy aware consolidation for cloud computing. In *Proceedings of the 2008 conference on Power aware computing and systems*, volume 10, pages 1–5. San Diego, California, 2008.

[129] Robert N Stavins. Experience with market-based environmental policy instruments. *Handbook of environmental economics*, 1:355–435, 2003.

[130] János Sztrik. Basic queueing theory. *University of Debrecen, Faculty of Informatics*, 193, 2012.

[131] Baya Takhedmit and Karim Abbas. A parametric uncertainty analysis method for queues with vacations. *Journal of Computational and Applied Mathematics*, 312:143–155, 2017.

[132] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629, 2009.

[133] Kishor S Trivedi. *Probability & statistics with reliability, queuing and computer science applications*. John Wiley & Sons, 2008.

[134] Catia Trubiani, Indika Meedeniya, Vittorio Cortellessa, Aldeida Aleti, and Lars Grunske. Model-based performance analysis of software architectures under uncertainty. In *Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures*, pages 69–78. ACM, 2013.

[135] Dean M Tullsen, Susan J Eggers, and Henry M Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *ACM SIGARCH Computer Architecture News*, volume 23, pages 392–403. ACM, 1995.

[136] Ward Van Heddeghem, Sofie Lambert, Bart Lannoo, Didier Colle, Mario Pickavet, and Piet Demeester. Trends in worldwide ict electricity consumption from 2007 to 2012. *Computer Communications*, 50:64–76, 2014.

[137] András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[138] Arunchandar Vasan, Anand Sivasubramaniam, Vikrant Shimpi, T Sivabalan, and Rajesh Subbiah. Worth their watts?-an empirical study of datacenter servers. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1–10. IEEE, 2010.

[139] Akshat Verma, Puneet Ahuja, and Anindya Neogi. pmapper: power and migration cost aware application placement in virtualized systems. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pages 243–264. Springer-Verlag New York, Inc., 2008.

[140] Lizhe Wang and Samee U Khan. Review of performance metrics for green data centers: a taxonomy study. *The journal of supercomputing*, 63(3):639–656, 2013.

[141] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced cpu energy. In *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, page 2. USENIX Association, 1994.

[142] Adam Wierman, Lachlan LH Andrew, and Ao Tang. Power-aware speed scaling in processor sharing systems. In *INFOCOM 2009, IEEE*, pages 2007–2015. IEEE, 2009.

[143] YunNi Xia, MengChu Zhou, Xin Luo, ShanChen Pang, and QingSheng Zhu. A stochastic approach to analysis of energy-aware dvs-enabled cloud datacenters. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(1):73–83, 2015.

[144] Yu Xiao, Pieter Simoens, Padmanabhan Pillai, Kiryong Ha, and Mahadev Satyanarayanan. Lowering the barriers to large-scale mobile crowdsensing. In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, page 9. ACM, 2013.

[145] Liang Yin, Ricardo M Fricks, and Kishor S Trivedi. Application of semi-markov process and ctmc to evaluation of ups system availability. In *Reliability and Maintainability Symposium, 2002. Proceedings. Annual*, pages 584–591. IEEE, 2002.

[146] Xiaoxiao Yu, Wenzhu Zhang, Lin Zhang, Victor OK Li, Jian Yuan, and Ilsun You. Understanding urban dynamics based on pervasive sensing: An experimental study on traffic density and air pollution. *Mathematical and Computer Modelling*, 58(5):1328–1339, 2013.

[147] Lotfi A Zadeh, King-Sun Fu, and Kokichi Tanaka. *Fuzzy sets and their applications to cognitive and decision processes*. Academic press, 2014.

[148] Matei Zaharia, Dhruba Borthakur, J Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Job scheduling for multi-user mapreduce clusters. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-55*, 2009.

[149] Yan Zhai, Xiao Zhang, Stephane Eranian, Lingjia Tang, and Jason Mars. Happy: Hyperthread-aware power profiling dynamically. In *USENIX Annual Technical Conference*, pages 211–217, 2014.

[150] Wen-Jun Zhang and Xiao-Feng Xie. Depso: hybrid particle swarm with differential evolution operator. In *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, volume 4, pages 3816–3821. IEEE, 2003.

[151] Xiaojing Zhang, Therese Lindberg, Krister Svensson, Valeriy Vyatkin, and Arash Mousavi. Power consumption modeling of data center it room with distributed air flow. *International Journal of Modeling and Optimization*, 6(1):33, 2016.