UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

Vers la modélisation de clusters de centres de données vertes

Dimitra POLITAKI

Université Côte d'Azur, Inria, CNRS, I3S

Présentée en vue de l'obtention du grade de docteur en Informatique **d'**Université Côte d'Azur

Dirigée par : Sara Alouf

Soutenue le : 16 juillet 2019

Devant le jury, composé de :
Sara Alouf , Chercheuse, Inria
Georges Da Costa, Associate Professor, IRIT, Université de Toulouse
Fabien Hermenier, Member of Technical Stuff, Nutanix
Vincenzo Mancuso, Research Associate Professor, IMDEA
Michel Riveill, Professeur, I3S/UNSA
Matteo Sereno, Professor, University of Turin

LABEX UCN User Centric Network @ SOPHIA

Inria INVENTEURS DU MONDE NUMÉRIQUE

cnrs

i3s sophia antipolis

On Modeling Green Data Center Clusters

Vers la modélisation de clusters de centres de données verts

Jury:

President du jury:
Michel Riveill, Professeur,Université Nice Sophia Antipolis

Rapporteurs :

Georges Da Costa, Associate Professor, IRIT, Université de Toulouse
Matteo Sereno, Professeur, Université de Torino

Invités:

Fabien Hermenier, Member of Technical Stuff, Nutanix
Vincenzo Mancuso, Research Associate Professor, IMDEA

Vers la modélisation de clusters de centres de données verts

Résumé: La consommation énergétique des clusters de centres de données augmente rapidement, ce qui en fait les consommateurs d'électricité à la croissance la plus rapide au monde. Les sources d'électricité renouvelables et en particulier l'énergie solaire en tant qu'énergie propre et abondante peuvent être utilisées pour couvrir leurs besoins en électricité et les rendre «verts», c'est-à-dire alimentés par le photovoltaïque. Ce potentiel peut être exploré en prévoyant l'irradiance solaire et en évaluant la capacité fournie pour les clusters de centres de données. Dans cette thèse, nous développons des modèles stochastiques pour l'énergie solaire; un à la surface de la Terre et un second qui modélise le courant de sortie photovoltaïque. Nous d'abord validons nos modèles par des données réels, puis nous proposons une étude comparative avec d'autres systèmes, notamment les modèles dits on-off. Nous concluons que notre modèle d'irradiance solaire peut capturer les corrélations multi-échelles de façon plus optimale, et il se montre particulièrement convénient dans le cas d'une production à petite échelle. De plus, nous proposons une nouvelle analyse de cycle de vie pour un système de cluster réel, ainsi qu'un modèle de cluster prenant en charge la soumission de travaux par lots et prenant en compte le comportement client impatient et persistant. Enfin, pour comprendre les caractéristiques essentielles du cluster d'ordinateurs, nous analysons deux cas: le complexe Google publié et le Nef cluster de l'Inria. Nous avons également implémenté marmoteCore-Q, un outil de simulation d'une famille de modèles de file d'attente, basé sur nos modèles.

Mots clé: L'énergie renouvelable, l'énergie solaire, le processus semi-markovien, la théorie des files d'attente, informatique en nuage, le cluster de centres de données, la caractérisation de la charge de travail, le cluster de Google, le cluster de Nef

On Modeling Green Data Center Clusters

Abstract: Data center clusters' energy consumption is rapidly increasing making them the fastest-growing consumers of electricity worldwide - expecting 13% of the total electricity world-wise in 2030. Renewable electricity sources and especially solar energy as a clean energy and abundant in many locations, can be used to cover their electricity needs and make them "green". In this thesis we develop stochastic models for solar energy; one at the surface of the Earth and a second one which models the photovoltaic output current. We then compare them with the state of the art on-off model and validate them with real data. Conclude the solar irradiance model can captures better the multiscales correlations and is suitable for small scale cases like ITC applications. We first propose a new job life-cycle of complex and real cluster system and then a multi-server model for data center clusters that supports batch job submissions and takes into account both impatient and insistent customers' behavior. To understand the essential computer cluster characteristics, we analyze in details two different workload type traces; the first one is the published complex Google trace and the second one which is simpler and serves scientific purposes, is from the Nef cluster located at the research center Inria Sophia Antipolis. To simulate our model and evaluate the data center clusters' performance, we first extract subtraces relative to a subset of the machines and use these for validation and comparison purposes. We then implement the marmoteCore-Q, a tool for the simulation of a family of queueing models based on our multi-server model for data center clusters with abandonments and resubmissions.

Keywords: Renewable energy, solar power, semi-Markov process, queueing theory, cloud computing, data center cluster, workload characterization, Google cluster, Nef cluster

To my spiritual father, priest Stefanos Poulis
and my mother Despoina ...
Στον πνευματικό μου πατέρα, πρωτοπρεσβύτερο Στέφανο Πουλή
και στη μητέρα μου Δέσποινα ...

Acknowledgments

I would like to express my gratitude to my advisors Sara Alouf and Fabien Hermenier for believing in me and offering me this opportunity, the freedom that they allowed me to find my own research path, their trust, and all of our discussions. A special thank you goes also to Alain Jean-Marie for all his valuable help, our fruitful discussions, guidance, and time.

I would like to express my appreciation to the members of my doctoral committee: Georges Da Costa, Vincenzo Mancuso, Michel Riveill, and Matteo Sereno. Thank you for accepting our invitation, reading my dissertation with eagerness, and for your useful feedback. It was a pleasure interacting with all of you. Special thanks to Georges and Matteo for their thorough review of my thesis and their valuable comments and suggestions. A huge thanks also to Matteo for inviting me to Turin and making my visit there really productive and enjoyable.

I am deeply indebted to Mr. Jean-Claude Bermond who is the person why I pursued my postgraduate studies in France. He selected me among many candidates for my master scholarship Vrika! and provided me with useful advice.

Thanks to Anne Combe, Marc Vesin, and Fabrice Huet; indeed, part of the analysis in *Chapter 4* were possible thank to Anne (Data Protection Officer at Inria) who provided us the Nef traces based on the General Data Protection Regulation (G.D.P.R) and Marc (IT staff at Inria, Sophia Antipolis) who kindly collected the Nef cluster traces, and Fabrice for his help in the first steps of pre-processing the Google traces. Additionally, I would like to thank András Horváth and Gábor Horváth for their support to the BMAP and PhFit tools respectively and their insights into the field.

A great many thanks to the scientist, my professor and my mentor in Patra, Athanasios Tsakalidis who provides me with valuable advice - both in life and research - throughout the years I know him. Σας ευχαριστώ δάσκαλε! Thanks to my professor Christos Bouras and all the members of the Research Unit 6 in Computer Technology Institute and Press - Diophantus, Patras, Greece for introducing me to the research world and which gave me a motivation to pursue a Ph.D. A warm thank you is also owed to Evi Papaioannou who most willingly offered me her help when needed.

My sincere thanks to all people from NEO and Scale teams that welcomed me and made me feel at home. I convey my gratitude to Giovanni Neglia for creating the opportunity to visit Matteo Sereno in Turin, which enhanced my thesis. Special thanks to Kostia Avrachenkov for ''to the point" scientific discussions and his encouragement. A huge thanks to Eitan Altman who was among the first ones who recognized the maturity of my work. Thanks to my colleagues and friends for the nice moments: Arun, Tania, Kostas, Berksan, Chuan, Guilhermo, Glib, Sasha, Foivos, Alina, Sophie, Maximilien, Vincenzo, and Vincent. Special thanks to my excellent, likeminded friend Abhishek Bose for walking alongside me down the academic path and supported me all the way. Thanks to my friend Nicolas Allegra for the enlightening discussions, which help me see my professional future with another perspective. Special thanks also to my friend Eleni Vatamidou for sharing her knowledge kindly with me. A huge thanks to Vera Samoili for standing by me during the last period of my Ph.D. A huge thanks also to NEO assistant and my friend, Laurie for all her valuable help with UCA and not only matters. Thanks to Ephie Deriche for our long French-Greek discussions.

A huge thanks to the close friends formed during the year of our Ph.D lives. Naoufal, I enjoyed a true friendship with you all these years. Hardik, you are an example for all of us,

thanks for being my friend. Konstantinos, I consider you as a member of my family, thanks for your genuine concern about me and endless support all these years. Eleni (Firippi), for understanding and supporting me. Karyna and Osama that became true friends having endless deep conversations. A huge thank to my best friends from Greece: Dimitra, Vivi, Damianos and Ariadni for their company and support all these years. My sincere thanks to all ubineters friends (promo 2014–2015) for this amazing multicultural and full of knowledge journey. Special thanks to Alexandros whom I consider as my elder brother, "the voice of logic", and for being around in various occasions.

Thanks to my colleagues and friends from the PhD-Seminars Inria-SAM and the "Le MOnde des Mathématique Industrielles" (MOMI) committees: Christos, Konstantinos, Eleni, Jenny, Nathalie, Selma, Milica, Dora, Isa, Michael and Ali. Thanks for this nice work! Special thanks to M. Denis Talay for his support to initiate these events. Next, I would like to thank my colleagues from "Mediation et Animation scientifiques Inria Sophia Antipolis – Méditerranée" for the nice collaboration during the scientific dissemination events.

Last but most importantly, I'd like to thank my family for their never-ending love and support: τους γονείς που με γαλούχησαν να είμαι ελεύθερη και μου έδωσαν τα εφόδια ώστε να επιλέξω να κάνω στη ζωή μου αυτό που επιθυμώ, που με δίδαξαν οτι η σκληρή δουλειά και η επιμονή οδηγούν στην επιτυχία, το μικρό μου αδελφό Μιχάλη που είναι πάντα δίπλα μου, τη γιαγιά μου Αργυρώ που μου δίδαξε οτι η υπομονή είναι από τις μεγαλύτερες αρετές, τον λατρεμένο μου παππού Μιχάλη για τη σοφία και την εμπειρία ζωής που μου έχει μεταλαμπαδεύσει και που φροντίζει πάντα, πριν από εμάς για εμάς.

This thesis is dedicated to my mother Despoina who taught me from an early stage of my life to be responsible, independent, work hard, never give up, and make good use of my time. But primarly, I dedicate this Ph.D thesis to my spiritual father, priest Stefanos Poulis. This manuscript is essentially completed successfully due to his great faith in God and his endless support whenever I was lacking courage. Thank you for counseling, encouraging, protecting, teaching, and believing in me. Αφιερώνω αυτή τη διδακτορική διατριβή στον πνευματικό μου πατερά πρωτοπρεσβύτερο Στέφανο, η οποία ουσιαστικά του ανήκει διότι χωρίς εκείνον και τη μεγάλη του πίστη στο Θεό δε θα τα είχα καταφέρει. Σας ευχαριστώ πατέρα μου, που με συμβουλεύετε, με στηρίζετε, με προστατεύετε, με διδάσκετε και πιστεύετε σε μένα ακόμη και τις στιγμές που χάνω το κουράγιο μου.

Dimitra POLITAKI
politakidimitra@gmail.com
Sophia Antipolis, France

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A data center cluster or simply a data center can be defined as a room of a building, or a whole building or a group of buildings, which houses a large group of networked computers called "servers". Data center clusters are set up and used by organizations and their clients (e.g., Google, Facebook, etc.) for the remote storage, processing, or distribution of large amounts of data. Some of them consume a lot of electricity power equivalent to that of a small city. Renewable electricity sources can been used to cover their electricity needs and reduce environmental footprint. In this thesis, we call this setup a green data center cluster. The analysis and modeling of green data center clusters is an interdisciplinary academic field at the intersection of physics, mathematics and computer science.

The idea of data centers has its roots in the Electronic Numerical Integrator and Computer (ENIAC), a huge computer room, established in 1945 in Philadelphia, Pennsylvania, U.S for military purposes. In 1960, data centers were mainframe computers that would fill up an entire room. They were costly and businesses rent space on the mainframe to perform certain functions. By the 1980s, the microcomputer industry was booming, and more and more companies established computers in the office. In the 1990s, the dot-com bubble leads to the boom of data center constructions. Suddenly, every organization needed fast Internet and Web servers. Small companies could not afford to set up such equipment, but managed cloud computing companies like Rackspace [1] started to build co-location facilities, housing thousands of servers and built up data centers. Data center power consumption became one of the main concerns of the community.

Due to the fast development of the data center clusters in the 21st century, the increase in data center cluster power consumption is inevitable. Data centers are growing continuously in size, complexity and energy density due to the storage demands, networking and computation. This has led to a worldwide energy problem. In [S.V. Garimella 2013], it is mentioned that data center clusters use 1.5% of the total electricity world-wise in 2010 and in [Andrae 2015], it is expected that they will take up 3-13% of global electricity in 2030. One of the reasons of this increase is the rapid annual growth rate to around 25% for global data center cluster usage. Even though, the electricity usage is unlikely to be reduced, an effective solution to that can be found in the renewable energy sources [Íñigo Goiri 2015, Oró 2015]. Solar energy is a promising clean energy technology which is abundant in many locations. Another reason that solar energy is preferable to feed data center cluster, is the low cost; the lifetime of a photovoltaic panel can surpass the 25 year and solar costs continue to decrease rapidly year after year and tend to be cheaper than brown energy. Brown energy prices are higher during the day, which is the on-peak period for data center clusters and during the night, the off-peak period. On the other hand, the amount of solar energy is higher during the day and non-existent during the night. Thus, the sun can cover the data center cluster electricity needs

[1] https://www.rackspace.com/

by feeding their solar panels during this time when the energy provided is at its maximum.

During the past decades, the demand for resources and capacity of data centers due to the dynamic nature of modern business have exploded. Companies now have to support a continuously increased amount of data, since more and more people use data centers' services. One of the major concerns of the modern companies is the reliable access to their data, thus data centers cannot take the risk of interruptions or running out of capacity. To understand and optimize the running cost of data centers and the ability to cover their actual and future needs, accurate capacity planning is vital. It implies that models for data center workload prediction are required and thus, a detailed analysis of workload characterization is necessary. For this reason, Google publishes a 29-day dataset seven years ago which has attracted the researchers' interest. We can conclude that predicting data center workload does not only facilitate data center computing companies but also creates a valuable source of information for researchers and business.

1.1 Why Study Solar Power?

With the increase in population and the boom of technology, the electricity needs are expected to grow rapidly year by year. Hence, it is hard to deny that renewable energy sources, among them solar power, are playing an important role in ecosystem and consequently in lives of people. According to the National Renewable Energy Laboratory [2], sunlight received by earth in one hour is enough to cover the energy needs per year of all people around the world. In the following, we outline the reasons why we believe solar power should be studied.

Diverse Applications. Solar energy can create photovoltaic power (PV) or concentrated solar power (CSP) for solar heating and can be used for diverse purposes. The most common are solar transportation, solar tech such as data centers; one of the most electricity high consumer in the world, solar heating. Moreover, solar power can be used for electricity production in the regions that access to the energy grid is limited, to distill water in areas with limited clean water supplies and even to power satellites in space.

Ecological Benefits. Solar can replace current fossil fuels like oil, coal and natural gas. According to the World Wide Fund For Nature [3], the electricity, which is generated by fossil fuels, causes harmful gas emissions that affect the air, water and soil pollution. Therefore, the global temperature is rising and climate weather patterns are changing. On the other hand, solar energy is environmentally friendly. Sun is an unlimited source of energy that prevents destruction of habitats, landscape damages and avoids harming the ozone layer. Solar energy is a clean and safe energy source, which is a future investment of our planet and protects the environment for the next generations.

Economic Benefits. Based on Renewable Power Generation Costs in 2017 report of International Renewable Energy Agency (IRENA) [4], the cost of renewable energy sources is decreasing that it will be less than that one of the traditional fossil fuels by 2020. Since

[2] https://www.nrel.gov/
[3] https://www.worldwildlife.org/
[4] https://www.irena.org/

2010, the cost of solar photovoltaic (PV) electricity has fallen by 73%. Specifically, the cost of solar PV is down to $0.10 per KwH, whereas the cost of electricity generated by fossil fuels is on average $0.11 per KwH. Moreover, panels require almost no maintenance and most reliable solar panel manufacturers offer 20-25 years warranty. The fact that solar energy provides affordable electricity prices across the countries can increase the market competition.

Social Benefits. Solar power can improve public health. The use of fossil fuels for energy, can lead to breathing problems, neurological damage, cancer, heart attacks, premature death, and other serious healthy problems. However, solar system are environmentally friendly as already mentioned and the above negative health impact can be avoided. Furthermore, thank to the development of solar energy, thousands of jobs have been created not only in solar industry but also in research domains, since many funding for innovations are devoted to the solar power.

1.2 Challenges and Contributions

In this section, we describe the main steps towards modeling real data center clusters fed by solar energy (or green data center clusters), the challenges we faced, and the contributions we made. Our analysis consists of four main parts. (i) we want to understand how to model the solar power in order to predict the electricity amount at a small time scale, (ii) we propose a multi-server queuing model for real data center clusters (iii) we characterize the workload of two different data center clusters to extract our model's parameter (iv) we develop a queueing tool to simulate our complex queuing model.

1.2.1 Solar Power

To model solar power, we first need to understand how the sunlight reaches the surface of Earth and which are the factors that can influence the percentage that can be absorbed or be lost on the way. Actually, it is obvious that there are some astrological explanations and weather effects that are responsible for that. For example, not all the regions all over the world receive the same amount of solar radiation the same period of the year. Another issue that should be addressed is the equipment that is necessary to convert solar energy to current. We faced the following challenges during this study.

1.2.1.1 Challenges

Solar pattern. Solar radiation corresponds to a non-periodic pattern. It varies during the day and from day to day. Radiation is zero during the night and mostly maximum around noon. It also increases from sunrise to noon and decreases from noon to sunset. However, solar radiation is not only irregular during the day but also during the seasons and during the years.

Solar variations. For the same geographical location, the solar radiation variations due to climate and seasons are expected. These variations can be observed from year to year because of the changes of climatic conditions. There are also variations from one season to

another season because the climate factors such as clouds, humidity, etc are changed. For instance, as we expect at the same place, in the winter there are in general more clouds than in the summer, consequently more solar radiation to the ground. Even though, there are sunny days during winter and cloudy days during summer.

System model. The choice of the system model to generate power by a PV panel, needs interdisciplinary knowledge. Questions like which astronomical model should be chosen to take the effective solar radiation, which photovoltaic panel module should be answered. In brief, the general idea is to collect solar radiation data and apply an astronomical model to take the effective solar radiation that hits the photovoltaic panels. We then convert the effective solar radiation to electrical power and we finally use a DC/DC power processor to maximize it. In this thesis, we focus on the current at the output of power processor. Details on how this current is computed, can be found in Section 2.2.3.

1.2.1.2 Contributions

We first develop a stochastic model for the solar irradiance which is defined as the product of irradiance obtained in clear sky and the disturbance due to climatic conditions; these two components are modeled separately. We then propose a stochastic on-off model for the current that comes out of photovoltaic panels. We compare this model to the on-off power source model developed by Miozzo et al. [Miozzo 2014]. In our on-off model, the output current is frequently resampled instead of being a constant during the duration of the "on" state as proposed in [Miozzo 2014].

We have collected and processed per-minute solar irradiance data from the National Renewable Energy Laboratory to validate and compare the proposed models, capturing small time scales fluctuations. Next, we compare the autocorrelation functions for all proposed models and the clear sky index model captures the multiscale correlations that are presented in the solar irradiance. Finally, we test the power spectrum density of generated trajectories of clear sky index model which is close to the measured one.

1.2.2 Modeling real data center clusters

Given the complexity of real data center clusters, it is necessary to understand their features; how the customers arrive to the system, how the customers are served by the system, how many servers are used; are there impatient customers or happy customers that can resubmit their jobs after execution completion. During this analysis we faced the following challenges.

1.2.2.1 Challenges

Modeling problems. There are many challenges faced when modeling real data center clusters. Jobs that arrive to the system, can consist of more than one task. Actually, based on TORQUE [Staples 2006] and OAR [Capit 2005], the state of the art open-source schedulers used in thousands clusters, jobs can be submitted in batches with specific commands and also resubmitted (e.g, best effort jobs (see Section 4.2.2)). On the other hand, the majority of queueing models in the literature [Wolff 1989, Harchol-Balter 2013] assume for simplicity single customer arrival to the system. It implies that any job consists

of one task. Another common feature in queueing theory, is the exponential assumptions both in arrival and service process, which are unrealistic but necessary for mathematical tractability. However, recent works dedicated to the diversity of different types of workload characterization such as [Amvrosiadis 2018] show that the Poisson assumptions for the above processes do not fit to the real data center clusters. Another important feature that we should take into account for our model, is the impatient customers that abandon the system before being served. Hence, to evaluate the real data center cluster performance metrics, we should find another way to model these processes, considering all the above features as well.

Technical challenges. Another problem that appears when we try to simulate our queueing model is the computation of the stationary distribution. Due to the complexity of our model, the stationary distribution needs to be estimated numerically. Our model represents a Markov chain, with the dimension of its state space depending on the number of servers; this number can get very large in a real data center cluster. The service process can described as a c-dimensional process, where c is the number of servers, checking if each server is busy or not. Thus, this dimension of the state space is prohibitive, size-wise, for any computer memory, rendering the stationary distribution of the system impossible to calculate. In the case it can be calculated, however, we need a tool that produces correct results in a fast manner.

1.2.2.2 Contributions

We first propose a new job/task life-cycle inside the data center cluster, which fits to the features of today's system schedulers [Capit 2005, Staples 2006]. We then develop a generic multi-server queue model describing the evolution of the number of tasks in a data center cluster. The arrival process is determined by a Batch Markovian Arrival Process (BMAP) and the service time has a phase type (PH-type) distribution. Unhappy customers may abandon the system and happy ones can be resubmitted to the system. As far as the problem of the computation of the stationary distribution as mentioned above, is concerned, we decrease the dimension of the state space of the service process, counting how many servers are in particular phase of service time distribution each time (or at time t). Thus, as showed in Section 3.3 we have found a way to decrease memory consumption. More details are presented in Section 3.3.

1.2.3 Workload Characterization

To model real data center clusters, it is hard to deny that the workload characterization plays a crucial role. Characterizing data center clusters workload is a very demanding task given the data center cluster complexity, the different types of workload and the hard-to-predict users' needs and behaviour. We choose to analysis the Google and Nef traces; two different workload types; the first one is the well-know complex Google trace which is available online [5] and the Nef one, collected from Inria Sophia-Antipolis Méditerrannée's data center cluster, is simpler and serves scientific purpose. During this analysis, we faced the following challenges.

[5] https://github.com/google/cluster-data/blob/master/ClusterData2011_2.md

1.2.3.1 Challenges

Data collection. The Google data center cluster workload dataset is available online [6]and it was downloaded following the documentation instructions. As far as the Nef cluster dataset is concerned, however, we encountered difficulties due to the General Data Protection Regulation (GDPR) that was implemented in Europe on 25th May 2018. The Nef data cluster serves both academic and scientific purposes and this is why the regulation on data protection and privacy becomes more severe aiming at protecting the users' individuality and scientific copyrights. Approximately four months were required to obtain permission to access these traces in a way that the Nef users' privacy is guaranteed.

Data processing. Another challenge is to process the data of a big size; the size of the compressed Google trace is approximately 41GB. The Google cluster data consists of six tables recording machines' attributes and events, jobs' events, tasks' events, usage and constraints. The model for real data center clusters that we developed in Chapter 3, is at the task level, therefore, most of our trace analysis is related to the task event table.

We downloaded the task event table for the 29-days period which is publicly available. The uncompressed information of task event tables is approximately 15.5GB large and is split in 500 different comma-separated values (csv) files. Each task event table line corresponds to a single event (e.g submit, schedule, finish, fail, etc) and consists of 13 fields. Each field contains information related to the event such as job index, task index, event type, etc. Each transition of the life-cycle of a task (see Section 3.2) corresponds to one event/record. Jobs arrive in the system in batches (by group). Approximately 25 % of the Google traces jobs consist of more than one task. Therefore, the problem is that not all the transitions for each job are included in the same csv file and this make more difficult the analysis of the trace. Even for one task, the transitions sometimes are not appeared in the same file. Hence, parsing all files for each job/task could take 1 or 2 days.

In order to process the data in detail and extract all the essential information for our model, we need to find a fast and efficient way to analyze them. This step is not straightforward due to two facts. First, due to the size of the dataset, it is important to have a fast access to the data. As mentioned before, not all the information for one job/task is included in one csv file. Hence, parsing all files to extract these information seems to be inevitable each time for a specific job/task, especially that we do not have any clue in which file the last job/task record is. The possibility of job/task resubmissions makes things more complex. For example, if a job/task has reached to the `finished` state of the life-cycle of a job/task, we could have stopped to look into the following csv files. However as it turns out that tasks are sometimes resubmitted, thus, we must continue to search for this particular job/task until the last line of the last csv file. Here is when the second fact comes into play due to the job/task resubmissions, it is important to keep copies in memory in order to avoid the repeated parsing/reading of the files and gain time for our following analysis, however disk memory is limited.

For this reason, at first, we merged the 500 csv files together, then sorted by job index and finally proceeded to our trace analysis. We have tested multiple state of the art tools

[6]https://github.com/google/cluster-data

including Matlab & Simulink, using vectorization [7] for taking exact results in a very fast way. By vectorizing the Matlab code, we achieve a shorter code without loops and, as a result, fewer programming errors with better performance. Another tool that we used, was the numPy Python library which is more complex in terms of code than the Matlab vectorized one and has overhead in terms of CPU utilization and in terms of memory consumption. In both cases, it took enough time to launch initially the 15.5 GB file and some results were suspicious.

1.2.3.2 Contributions

To understand the essential characteristics of a computing cluster for modelling purposes, we look into the two aforementioned datasets. After a preliminary analysis and sanitizing of each dataset, a numerical analysis is performed to characterize the different stochastic processes taking place in the computing cluster. In particular, we characterize the impatience process, the re-submission process, the arrival process (batch sizes and correlations) the waiting time and the service time, considering the impact of the scheduling class and of the execution type.

As we saw in the previous section, the conventional attempts at improving the code performance-wise fell through so, finally, we took the following steps in order to improve the code. First of all, all of the event information for the same job (job index) was put in the same file in the beginning of our data processing, using the Hadoop tool [8] and then we analyzed the traces using bash comments. More details are presented in Section 4.2.

Given the size of the Google computing center, it is not possible to find the stationary distribution of the queue parameterized for this center. To overcome the technical difficulties faced, as mentioned in Section 1.2.3.1 when validating the queueing model against the Google cluster data, we extract two subtraces relative to a subset of the machines and use these for validation and comparison purposes.

1.2.4 marmoteCore-Q tool

There are few environments for simulating queueing models that can allow programming a complex Markov model. The most popular mathematical modeling environment such as Matlab, Mathematica, Maple, R, etc provide packages with special functions to model and analyse queueing systems. We believe that there is the need of a tool, implemented in C/C++ such that to take the advantage of giving valid results in a short time, simulating even large-scale complex queueing models without analytical results. This tool can also be used to simulate simple queues with explicit closed-formulas. During this implementation, we faced the following challenges.

1.2.4.1 Challenges

To validate complex queueing models with no analytical expressions, it is a non-trivial task. Each queueing model can be represented by a Markov chain, which has an infinitesimal generator Q matrix. Each entry of the matrix Q corresponds to one Markov chain state and is represented by a vector. To find the vector given the position and to find the position

[7] https://fr.mathworks.com/help/matlab/matlab_prog/vectorization.html
[8] http://hadoop.apache.org/

given a vector are two implementation challenges. Details on how these technical challenges are overcome are presented in Section 5.1.3.

1.2.4.2 Contributions

Using the marmoteCore platform [Jean-Marie 2017], we have developed the marmoteCore-Q tool for the simulation of a family of queueing models based on the general $BMAP/PH/c$ queue with impatience and resubmissions. There exist many special cases of this queue for which analytical results are known. Examples are: the $M/M/1$ queue and its finite capacity version, the $M/M/c/K$ queue, the $M/PH/1$ and $M/PH/\infty$ queues, the $M^X/M/1$ and $M^X/M/\infty$ queues. Such examples are used to validate the implementation of the marmoteCore-Q tool. marmoteCore-Q provides to users an easy and efficient way to simulate their own queueing models following the instructions which are explained in Chapter 5.

1.3 Thesis outline

This thesis has the following structure. In Chapter 2, we propose two stochastic models; the first one focuses on solar radiance and the second model focuses on the electrical intensity obtained at the output of a photovoltaic panel. In Chapter 3, we first describe a new life-cycle of a task and then develop multi-server queueing model with abandonments and resubmissions for data center clusters. The arrival process is modeled as a Batch Markovian Arrival Process and the service time service time is determined by a Phase-Type-distribution. In Chapter 4, we present the detailed characterization of two different data center cluster workloads; the first one is the 29-day Google data set available in the web and the second one is the Nef cluster dataset from from the data center cluster of research center of French National Institute for Computer Science and Automation (Inria) in Sophia Antipolis. We then extract the model's parameters from the traces to capture the queueing model features, which presented in Chapter 3. Chapter 5 proposes the marmoteCore-Q, a tool for simulating simple and numerically complex queues. The implementation details and validation are presented. Chapter 6 concludes the thesis and discusses the future work.

Chapter 2

Solar Power

Contents

In this chapter, we develop a stochastic model for the solar power at the surface of the earth. We combine a deterministic model of the clear sky irradiance with a stochastic model for the so-called clear sky index to obtain a stochastic model for the actual irradiance hitting the surface of the earth. Our clear sky index model is a 4-state semi-Markov process where state durations and clear sky index values in each state have phase-type distributions. We use per-minute solar irradiance data to tune the model, hence we are able to capture small time scales fluctuations. We compare our model with the on-off power source model developed by Miozzo et al. [Miozzo 2014] for the power generated by photovoltaic panels, and to a modified version that we propose. In our on-off model the output current is frequently resampled instead of being a constant during the duration of the "on" state. Computing the autocorrelation functions for all proposed models, we find that the irradiance model surpasses the on-off models and it is able to capture the multiscale correlations that are inherently present in the solar irradiance. The power spectrum density of generated trajectories matches closely that of measurements. We believe our irradiance model can be used not only in the mathematical analysis of energy harvesting systems but also in their simulation.

Note: Part of the material presented in this chapter is published in [Politaki 2017].

2.1 Introduction

In the past decade, there has been an awareness rising concerning the energy cost and environmental footprint of the fastly growing Information and Communication Technology (ICT) sector. In [Van Heddeghem 2014] Van Heddeghem et al. assess how did the electricity consumption of the ICT sector evolved between 2007 and 2012. They report an increase in the relative share of ICT products and services (communication networks, personal computers and data centers, excluding TVs' set-top boxes and (smart)phones) in the total worldwide electricity consumption from about 3.9% in 2007 to 4.6% in 2012. Even though devices from new technologies are more energy efficient, this is outweighed by the fast growth in their numbers.

Among the most promising approaches recently pursued to reduce the environmental footprint of the ICT sector, we focus on the use of renewable energy sources and in particular solar energy. As photovoltaic panels are being used worldwide to power multiple components of the ICT sector, there is an increasing effort in the literature to consider the solar energy production when modeling computer and communication systems. For illustration purposes, we mention two recent papers modeling ICT systems involving renewable energy sources.

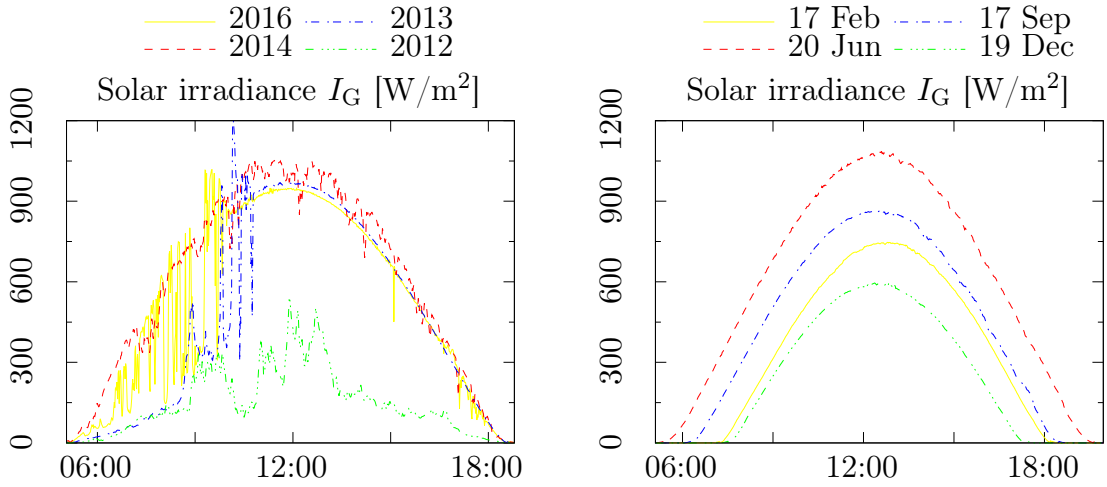
In [Dimitriou 2015], Dimitriou, Alouf and Jean-Marie consider a base station that is powered by renewable energy sources and evaluate in particular the depletion probability. The base station is modeled as a multi-queue queueing system where energy queues model the batteries that store the harvested energy. The authors of [Dimitriou 2015] model the renewable energy production as a Poisson process whose rate is modulated by a Markov chain representing the random environment.

Neglia, Sereno and Bianchi [Neglia 2016], consider the problem of geographical load balancing across data centers that have a dual power supply: grid and solar panels. They study the problem of scheduling jobs giving priority to data centers where renewable energy is available. The renewable energy source at each data center is modeled as an on-off process governed by a continuous time Markov chain. In the "on" state the data center can be fully powered by its renewable energy source; in the "off" state the data center is powered by the grid.

These examples among others illustrate the lack of a unified stochastic model for the solar energy to be used in the mathematical analysis of communication/computer systems. Our objective in this work is to develop such stochastic models for the solar power at the surface of the earth. Although there are a few models in the recent literature of the networking community [Miozzo 2014], these rely on per-hour measurements. Therefore, such models do not capture the fluctuations in the solar irradiance at smaller time scales.

Our main contribution combines a deterministic model of the *clear sky* irradiance with a stochastic model of the so-called *clear sky index* to obtain a model of the actual irradiance hitting the surface of the earth. We will compare our model (after converting the actual irradiance to power generated by photovoltaic panels) to the night-day clustering model developed by Miozzo et al. in [Miozzo 2014] for the generated power. Based on this work, we propose a modified night-day clustering model. Our model for the harvested power is that of an on-off source in which the power generated in each state is frequently resampled from an appropriate distribution capturing the short-time scale fluctuations observed in practice.

To evaluate our models, we consider the autocorrelation functions and the periodograms of



(a) May 1st, different years (b) Different days in 2012

Figure 2.1: Variations in the daily pattern of the solar irradiance are due to (a) the weather conditions and (b) the day of the year

the generated trajectories. The autocorrelation function illustrates how well do our proposed models capture the multiscale correlations found in the data, whereas the spectral analysis allows to determine which characteristic time-scales are reproduced by the models.

In order to build solar models for small time scales, we consider the following challenges arise; Beside the obvious dependency on the geographic location, the first challenge is that the solar irradiance exhibits a night-day pattern that is affected by weather conditions which may induce burstiness at multiple time scales. The second one is that the solar irradiance depends also on the day of the year.

We illustrate these variations in Fig. 2.1 where per- minute measurements of the solar irradiance in Los Angeles [Andreas 2012] are depicted for the same day of different years (Fig.2.1a) and for different days of the same year (Fig. 2.1b). Even with perfect weather conditions, there are differences between the solar irradiance on the same day of different years due to varying astronomical conditions.

The remainder of this chapter is structured as follows. We discuss the related work in Section 2.3. We then present, in Section 2.4 the model of the clear sky index, and the model of the generated power in Section 2.5. Finally, we assess our models in Section 2.6 and conclude in Section 2.7.

2.2 Background

For this chapter, we present the essential background to provide some intuition concerning the different types of solar irradiance, clear sky index and how the power is generated by a PV panel.

2.2.1 The Solar Irradiance

The amount of the solar energy that arrives per unit of time at a specific area of a surface is the solar irradiance and is expressed in W/m^2. We present and discuss below the different

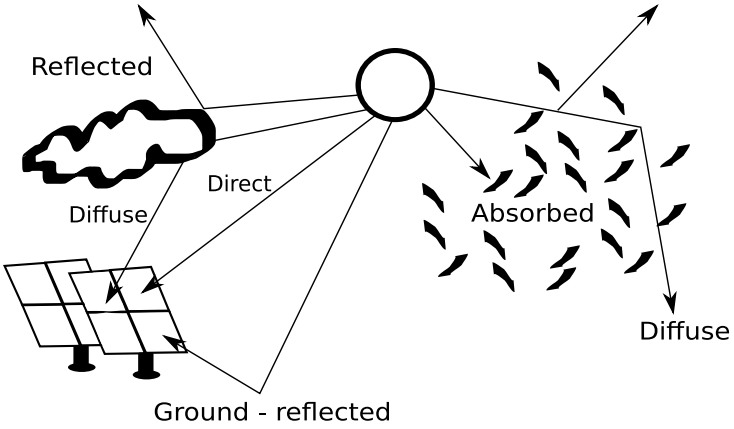


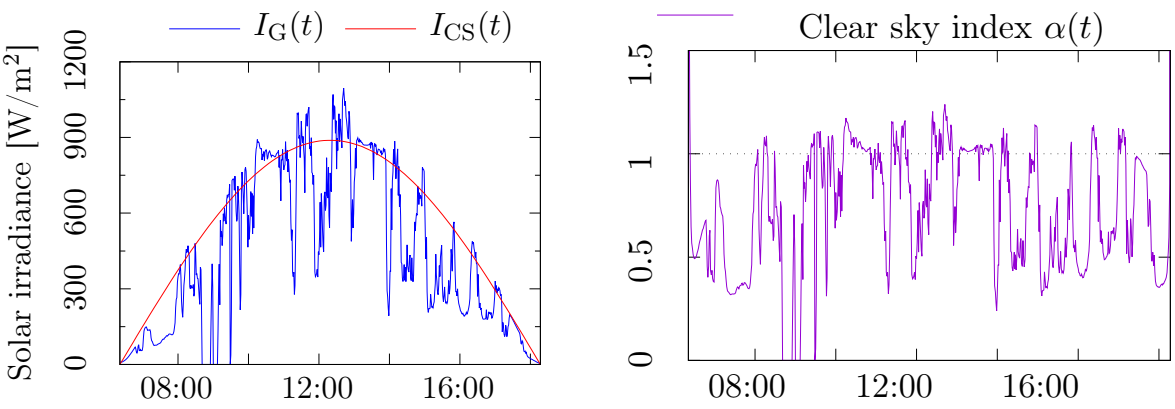
Figure 2.2: Different types of solar irradiance.

types of solar irradiance as illustrated in Fig. 2.2:

- *Absorbed Solar Irradiance* is the solar radiation which is absorbed by some atmospheric molecules, solar collectors, and the ocean. Absorbed radiation never reaches to the surface of the Earth.
- *Diffuse Normal Irradiance* (DNI) or diffuse is the radiation which strikes a point from the sky towards the surface of the Earth following a different angle. In case of clear sky, there should be almost no diffuse sky radiation, whereas in case of hazy atmosphere and/or cloud reflections DNI assumes high values.
- Direct Horizontal Irradiance (DHI) or direct is the amount of solar radiation from the direction of the sun.
- *Ground-reflected Irradiation* is the radiation from the sun which strikes the Earth and then is reflected back into the atmosphere.
- Reflected Irradiance is the irradiance which strikes a point from the sky and then travel upwards.

Global horizontal irradiance (GHI) or *global irradiance* is defined as the total solar radiation; the sum of direct horizal irradiance (DHI), diffuse normal irradiance (DNI) and ground-reflected irradiance. However, the ground-reflected irradiance is usually insignificant compared to direct and diffuse irradiance. For this reason, in [Andreas 2012], global radiation is the the sum of direct and diffuse radiation only and expresses as follow: $GHI = DHI + DNI * cos(Z)$, where Z is the solar zenith angle [1]. In the following the solar irradiance will refer to the *global* irradiance $I_{\mathrm{G}}(t)$ as defined above. The reason for this is that we will rely on daily measurements of the global irradiance [Andreas 2012] to tune

[1]Zenith Angle is the angle between the direction of the sun, and the zenith (directly overhead).



(a) Global and clear sky irradiances (b) Clear sky index $\alpha(t) = I_G(t)/I_{CS}(t)$

Figure 2.3: Illustrating the global irradiance $I_G(t)$, the clear sky irradiance model $I_{CS}(t)$ given in Eq. (2.2) and the resulting clear sky index $\alpha(t)$ on September 28th, 2010, in Phoenix, Arizona [f]

our models. No measurements of the ground-reflected radiations are available for download from [Andreas 2012].

We define below two meteorological parameters, the air temperature and the relative humidity, that influence the solar irradiance and will be used in the following.

- Air temperature is the weather parameter which is measured how hot or cold the air is. Temperature describes the gas kinetic energy that make up air. The air temperature decreases, when gas molecules move more slowly, whereas it increases. In the following, the air temperature is measured using the Celsius scale.

- Relative humidity is the amount of water vapor in the air. It expresses as the ratio between the measured amount and the maximum possible amount

2.2.2 Clear Sky Index

The solar irradiance $I_G(t)$ can be seen as the result of applying a multiplicative noise to the *clear sky* solar irradiance $I_{CS}(t)$. This multiplicative noise, denoted $\alpha(t)$ in this thesis and called *clear sky index* in the literature, captures the perturbations seen in the solar irradiance with respect to the clear sky solar irradiance. We have $I_G(t) = \alpha(t)I_{CS}(t)$. Figure 2.3 illustrates $I_G(t)$, $I_{CS}(t)$ and $\alpha(t)$ for a sample day.

2.2.3 The Power Generated by a PV Panel

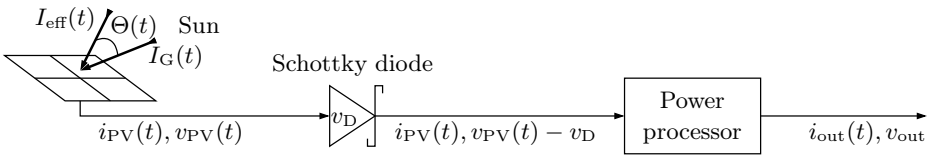


Figure 2.4: Using a fraction I_{eff} of the solar irradiance I_G, the PV cells generate a power (current i_{PV} and voltage v_{PV}) that goes through a Schottky diode and a power processor before it can be consumed

The solar irradiance can yield electricity through the use of a PV panel as shown in Fig. 2.4. The usable power is directly related to the solar irradiance arriving at the panel (that is I_G) as thoroughly explained in [Miozzo 2014] and implemented in the tool SolarStat that is available online [Gianfreda 2014]. The general idea is the following:

1. The solar irradiance effectively used by the PV panel is the component of $I_G(t)$ that is perpendicular to its surface, that is $I_{eff}(t)$. $I_{eff}(t)$ depends on $\Theta(t)$ which is the angle made by incident sunlight and the efficient irradiance to the PV panels.

2. The PV panel translates the effective solar irradiance I_{eff} into electric power with current $i_{PV}(t)$ and voltage $v_{PV}(t)$.

3. A Schottky diode reduces slightly the voltage but preserves the current.

4. A power processor extracts the maximum power from the PV panel and the output power has current $i_{out}(t)$ and voltage v_{out}.

The fluctuations seen in the solar irradiance $I_G(t)$ are still present in the output current $i_{out}(t)$. There may be additional fluctuations due to the local temperature and humidity that affect the functioning of the PV cells.

2.3 Related Work

Studies on the solar irradiance are abundant in the literature. Given the paramount role of the solar energy in many biological ecosystems, it is crucial to have models for the solar irradiance as measurements are not always available. For instance, Piedallu and Gégout develop in [Piedallu 2007] a model that can predict the accumulated solar energy anywhere, providing annual figures for an entire country, as would be required for predictive vegetation modeling at a large scale. However such biology-oriented models are not fit for ICT applications that evolve typically on a much smaller time scales than vegetation.

Targeting the design of a solar system, there is a large body of work focusing on the clear sky irradiance. To cite a few references, Dave, Halpern and Myers overview in [Dave 1975] several clear sky irradiance models and compare the accumulated daily and annual energy. They consider a tilted surface and account for both sky radiations and ground-reflected radiations. They find in particular that the effective irradiance at a surface is proportional to the cosine of the angle between the sunlight direction and the normal to the surface. Bird and Hulstrom compare in [Bird 1981] five models for the maximum clear sky solar irradiance and propose a sixth model based on algebraic expressions. All these models require many meteorological input parameters (e.g., the surface pressure, the total ozone, the precipitable water vapor).

Another important component when modeling the solar irradiance is the clear sky index. Jurado, Caridad and Ruiz characterize the clear sky index using 5-minute measurements of the solar irradiance [Jurado 1995]. They partition the data according to the solar angle, considering two one-hour intervals at a time (both intervals corresponding to the same range of solar angle). They find that the density of the clear sky index in each partition is bimodal and can be modeled as a mixture of Gaussian distributions. The parameters of the distributions and the mixing factor are obtained from measurements by least squares approximation. The

authors observe that the standard deviations of the Gaussian distributions depend on the solar angle. Also the bimodal behavior observed over 5-minute intervals is no longer observed when the interval in the data is larger. This is an important outcome that indicates that a model tuned with data having a given frequency of measurements can not match data having a different measurements rate. This observation supports our intuition that if one wants to use a model of solar power at a given time scale, then the model must be tuned with data at the same time scale. The authors of [Jurado 1995] are not clear on how do they compute the clear sky index from the measurements of the solar irradiance. Surprisingly, the computed clear sky index is always below 1 suggesting that they consider a very large maximum clear sky irradiance.

Gu et al. consider in [Gu 2001] a related metric which is the relative change of solar irradiance (this would be $100(\alpha - 1)$) under the impact of clouds. They analyze per-minute measurements of solar irradiance collected in Brazil over a period of two months during the wet season. They observe that broken cloud fields create a bimodal distribution for the relative change: shaded areas receive attenuated solar irradiance while sunlit areas may receive higher irradiance than under a clear sky. This effect is caused by radiations scattering and reflections from neighboring clouds. Conducting a spectral analysis on the time series of measured surface irradiance, they observe that clouds are responsible for two different regimes according to their types and density causing either large or small scale fluctuations. This study highlights the effect of clouds and have certainly impacted the development of subsequent models for the solar irradiance.

Miozzo et al. focus on the solar power generated by small embedded photovoltaic panels such as those used in sensor networks. They develop in [Miozzo 2014] two stochastic models in which the dynamics of the power source is described by a semi-Markov process with $N \geq 2$ states. The first model is an on-off power source and the authors tune the sojourn time and power in each state by using a night-day clustering on hourly measurements of the solar irradiance. In the second model, the power source goes through a number of N states in a round-robin way and all sojourn times are equal and constant. A time slot based clustering enables the authors to estimate the power distribution in each state.

Ghiassi-Farrokhfal et al. consider also the solar power generated by photovoltaic panels but in the context of dimensioning an energy storage system. To near-optimally size a storage system, they develop in [Ghiassi-Farrokhfal 2015] a new *envelope* model for the generated power. In the general envelope model, the solar power is characterized by a statistical sample path lower envelope such that the probability of having the maximum of the distance envelope-solar energy exceed a given value is upper bound by a characteristic bounding function evaluated at the given value. Inspired by the findings of [Gu 2001], the authors of [Ghiassi-Farrokhfal 2015] adapt the general envelope model to enable a separate characterization of the three underlying processes of solar power (diurnal, long-term, and short-term variations).

To sum up, even though there is ample bibliography related to solar energy models, the majority of the existing models have not been validated against real data; even the few models that have been compared to actual solar traces are only suitable for large scale applications. However, in state-of the art domains such as ITC applications, it is necessary to build solar models to cater to small scale cases as well.

2.4 Modeling the Solar Irradiance I_G

In this section, we focus on the solar irradiance $I_G(t)$. Our aim is to define a model able to capture the small time-scale fluctuations inherently present in the global irradiance. To that end, we model separately the clear sky irradiance $I_{CS}(t)$ and the clear sky index $\alpha(t)$. By definition, we have

$$I_G(t) = \alpha(t) I_{CS}(t) \ . \tag{2.1}$$

We discuss $I_{CS}(t)$ in Section 2.4.1 and model $\alpha(t)$ in Section 2.4.2.

2.4.1 Modeling the clear sky irradiance $I_{CS}(t)$

The solar irradiance arriving at a surface during a clear sky day without any perturbations due to a change in the meteorological conditions exhibits a predictable pattern as shown in Fig. 2.1b. The models discussed in [Dave 1975] for the hourly clear sky irradiance and in [Bird 1981] for the maximum clear sky irradiance are not easily applicable given the unavailability of many input parameters (e.g. the air mass, the carbon, dioxide and oxygen absorptance, aerosol measurements etc). Instead, we use the so-called "simple sky model" [Iqbal 1983] which defines a simple sinusoidal form for each day, taking into account the times of sunrise and sunset and the maximum clear sky irradiance. The clear sky irradiance $I_{CS}(t)$ is given by the following equation:

$$I_{CS}(t) = \text{MaxClearSky} \cdot \sin\left(\frac{t - \text{sunrise}}{\text{sunset} - \text{sunrise}}\pi\right) \ . \tag{2.2}$$

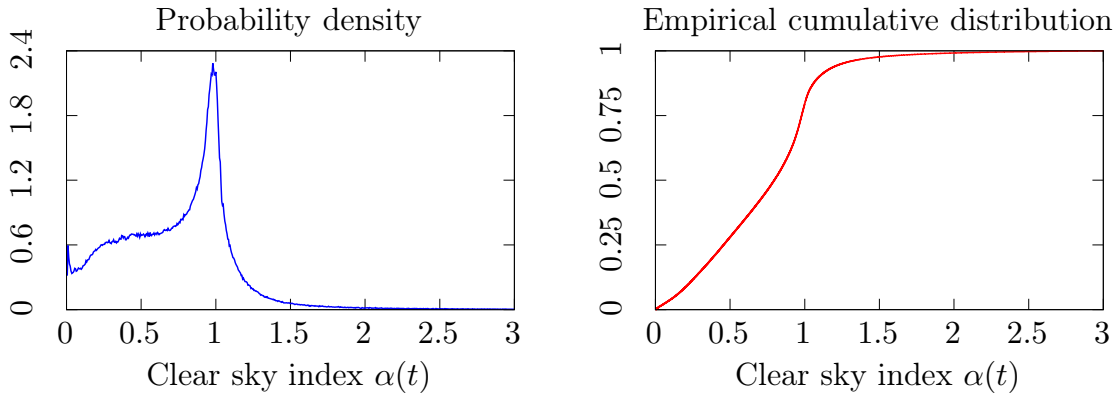
The values of "sunrise", "sunset" and "MaxClearSky" are astronomical data that can be easily obtained in practice for any date and many selected locations from the website [ptaff.ca] (the maximum clear sky irradiance is called there "maximal solar flux"). An illustration of Eq. (2.2) is in Fig. 2.3a.

2.4.2 Modeling the clear sky index $\alpha(t)$

The clear sky index $\alpha(t)$ captures the fluctuations over time of the global irradiance with respect to clear sky conditions, as illustrated in Fig. 2.3b for a sample day and a sample location. Consequently, one thinks of defining a state for each macro weather condition. Based on our review of the literature, we define four states for $\alpha(t)$ that correspond to: heavy clouds between the sun and the surface (very low values of $\alpha(t)$), medium to light clouds between the sun and the surface (values of $\alpha(t)$ around 0.6), clear sky (values of $\alpha(t)$ around 1), and high reflection and diffusion in the atmosphere (values of $\alpha(t)$ larger than 1). We assume all transitions between different states to be possible.

We propose to capture the dynamics of $\alpha(t)$ by a discrete-time semi-Markov process.[2] Our model works as follows. When the process $\alpha(t)$ enters a state i, it will remain there for a duration τ_i governed by a probability density function f_i. While in state i, the clear sky index $\alpha(t)$ behaves like $\alpha_i(t)$, a stochastic process with probability density function g_i. When

[2] Using a discrete-time Markov process does not yield satisfactory results as correlations are not described well.



(a) (b)

Figure 2.5: Density and cumulative distribution curves of the clear sky index $\alpha(t)$ computed using Eq. (2.2) and per-minute solar irradiance data [Andreas 2012]

the sojourn time τ_i expires, the process *changes* its state. The distributions f_i and g_i, for $i \in \{1, 2, 3, 4\}$ will be fitted to empirical distributions of the sojourn times and values of $\alpha(t)$.

Data collection: To tune our model of $\alpha(t)$ we use per-minute measurements of the solar irradiance $I_G(t)$. We collect the data from National Renewable Energy Laboratory (NREL) [Andreas 2012] for the region of Los Angeles from April 2010 until March 2015. We select from here the following information:

- Start date
- End date
- Global Horizontal irradiance (W/m^2)
- Air Temperature (°C)
- Relative Humidity (%)

We then download the data, choosing selected 1-min data (ZIP compressed). Solar irradiance measurements are obviously zero during the night and thus, are not included.

We compute $\alpha(t) = I_G(t)/I_{CS}(t)$ using the data and Eq. (2.2) for each minute during the five years. We observe that we may well have in the real measurements $I_G(t) > 0$ around sunset and sunrise due to diffusion. As $I_{CS}(t) = 0$ at sunrise (and before) and sunset (and after), this implies that infinite values for the ratio $I_G(t)/I_{CS}(t)$ can occur. To discard such values when computing $\alpha(t)$, we enforce the (arbitrary) bound $\alpha(t) < 3$. For illustration purposes, we compute the density and the cumulative distribution of the clear sky index and depict them in Fig. 2.5.

The density of the clear sky index depicted in Fig. 2.5a is not bimodal as found in [Jurado 1995]. The measurements used in [Jurado 1995] were made every 5 minutes and the densities were computed over two intervals of 1 hour each corresponding to the same range of the solar angle. Instead, the density shown in Fig. 2.5a is for all 1-minute measurements over a period of 5 years.

Once that we have computed the values of $\alpha(t)$, we first aim to validate the number of states of our semi-Markov model. We apply the k-means clustering algorithm as it was

Table 2.1: Evaluation of the quality of the clustering for several values of k

k	Davis-Bouldin index	k	Davis-Bouldin index	k	Davis-Bouldin index
2	0.5113	5	0.5290	8	0.5072
3	0.5057	6	0.5059	9	0.5120
4	**0.5017**	7	0.5205	10	0.5070

Table 2.2: Values in each cluster according to k-means, their corresponding state in the semi-Markov model and weather condition

Range of values of $\alpha(t)$	State	Physical interpretation
$[0, 0.44152)$	1	heavy clouds between the sun and the surface
$[0.44152, 0.81639)$	2	medium to light clouds between the sun and the surface
$[0.81639, 1.4343)$	3	clear sky
$[1.4343, 3)$	4	high reflection and diffusion in the atmosphere

first introduced in [Kanungo 2002] and, particularly, the implementation in the core Matlab library. For a brief overview of the algorithm, one can refer to Appendix A. We use the Davies-Bouldin index to define the optimal number of clusters. The Davies-Bouldin index is based on a ratio of within-cluster and between-cluster distances. The smaller its value the better the clustering.

We tested nine different clustering (for $k \in \{2, \ldots, 10\}$) and computed the Davies-Bouldin index for each clustering obtained. The values of the index were between 0.5017 and 0.5290. The smallest value was obtained for $k = 4$ implying that ideally the values of $\alpha(t)$ should be classified into four clusters. This analysis supports our choice of having four states in the model for the clear sky index and each state is mapped to one of the four clusters obtained. The details on the four clusters/states obtained when applying the k-means clustering algorithm are given in Table 2.2.

We report the values of the Davies-Bouldin index in Table 2.1 for several values of k. We can make two observations: first, the value of the index does not vary much with the number of clusters k; and second, very close index values correspond to distant values of k (e.g. for $k = 3$ and $k = 6$). Nevertheless, the value $k = 4$ yields the smallest index hence the optimal is to split the data into four clusters. This analysis confirms our choice of having four states in the model for the clear sky index. Each state is mapped to one of the clusters as specified in Table 2.2.

Now that we have clearly identified the four states of our semi-Markov model, our next step is to identify the transition probabilities among the states. We estimate them using the computed values of $\alpha(t)$ and the identified clusters. We first map each computed value of $\alpha(t)$ to its corresponding state, then we count the number of transitions between any pair of states. The transition probability from state i to state j is estimated as the ratio of the number of transitions from state i to state j to the total number of transitions out of state i. We find the following transition probability matrix for the four-state semi-Markov model:

$$\boldsymbol{P} = \begin{bmatrix} 0 & 0.8361 & 0.0549 & 0.1090 \\ 0.3645 & 0 & 0.6296 & 0.0059 \\ 0.0274 & 0.9019 & 0 & 0.0707 \\ 0.0484 & 0.0536 & 0.8980 & 0 \end{bmatrix}. \tag{2.3}$$

Table 2.3: Number of phases of the phase-type distribution fitting the (shifted) sojourn times and values in each state

Variable	Number of samples used in the fitting	Number of phases of the phase-type distribution fitting the variable
$\tau_1 - 1$	19678	5
$\tau_2 - 1$	8456	6
$\tau_3 - 1$	2094	6
$\tau_4 - 1$	15400	6
$\alpha_1(t)$	298141	20
$\alpha_2(t) - 0.44152$	345973	20
$\alpha_3(t) - 0.81639$	563411	6
$\alpha_4(t) - 1.4343$	34432	3

As a final step, we aim is to characterize the densities f_i and g_i for $i = 1, \ldots, 4$. We carry out a statistical analysis on the computed values of α in order to determine the distributions of the sojourn times $\{\tau_i\}_{i=1..4}$ and the values $\{\alpha_i\}_{i=1..4}$. Observe that the sojourn time τ_i in a given state i corresponds to the number of consecutive values of α inside the corresponding cluster. Recall that α is a discrete-time process and as the measurements used for tuning the model are minute-based, then the time unit in our model is the minute. The sojourn times $\{\tau_i\}_{i=1..4}$ and the values $\{\alpha_i\}_{i=1..4}$ are bounded variables. After applying an appropriate shift to each of these random variables, we find the (truncated) phase-type distribution that best fits each one of the empirical distributions using the PhFit tool [Horváth 2002].

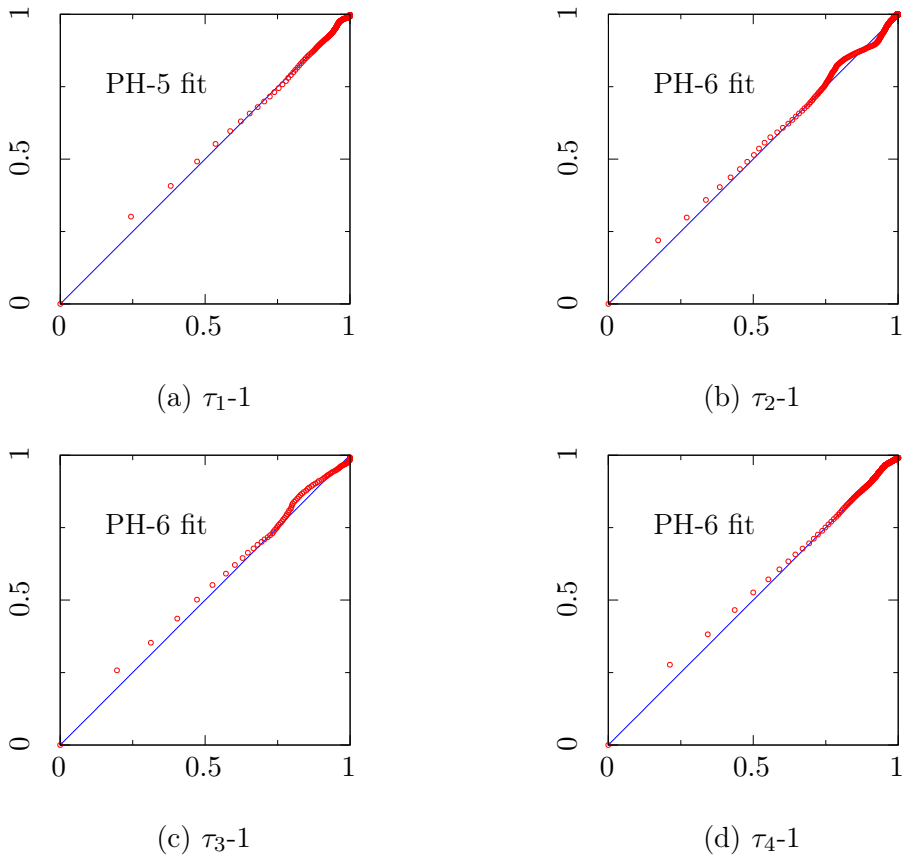
We choose the relative entropy [Cover 1991] as distance measure according to which the fitting is performed. To decide for the number of phases, we proceed in two steps: first, we identify the interval where the density is the highest; then, we choose the smallest number of phases that yields small relative error between the original and fitted cumulative distributions curves in the identified interval. The resulting fitted distributions are reported in Table 2.3.

To assess the quality of the fit, we use probability plots. Each graph in Fig. 2.6 depicts on the y-axis the probabilities of the fitted distribution against the probabilities of the sojourn times in a given state on the x-axis. Similar probability plots are displayed in Fig. 2.7 for the values of α in each state. We observe that the phase-type distribution fits reasonably well all variables for all states. We choose to proceed with this fitting nevertheless and see how the overall model performs before considering to use other distributions to fit the data.

Regarding the values of $\alpha(t)$ in each state, we can see in Fig. 2.7 that the selected phase-type distributions fit very well the values of $\alpha(t)$. We observe that the quality of fit for $\alpha_1(t)$ and $\alpha_2(t)$ is obtained at the cost of having a significantly larger number of phases (that is 20; see Table 2.3) with respect to the other variables.

2.5 Modeling the Harvested Power

To account for the power generated by PV panels when evaluating solar-powered systems, one has mainly two options. The first option is to use a model for the solar irradiance such as the one developed in Section 2.4 and then infer the power generated by the PV cells (or equivalently $i_{\text{out}}(t)$; see Fig. 2.4). This second step may be a simple linear model (i.e. the



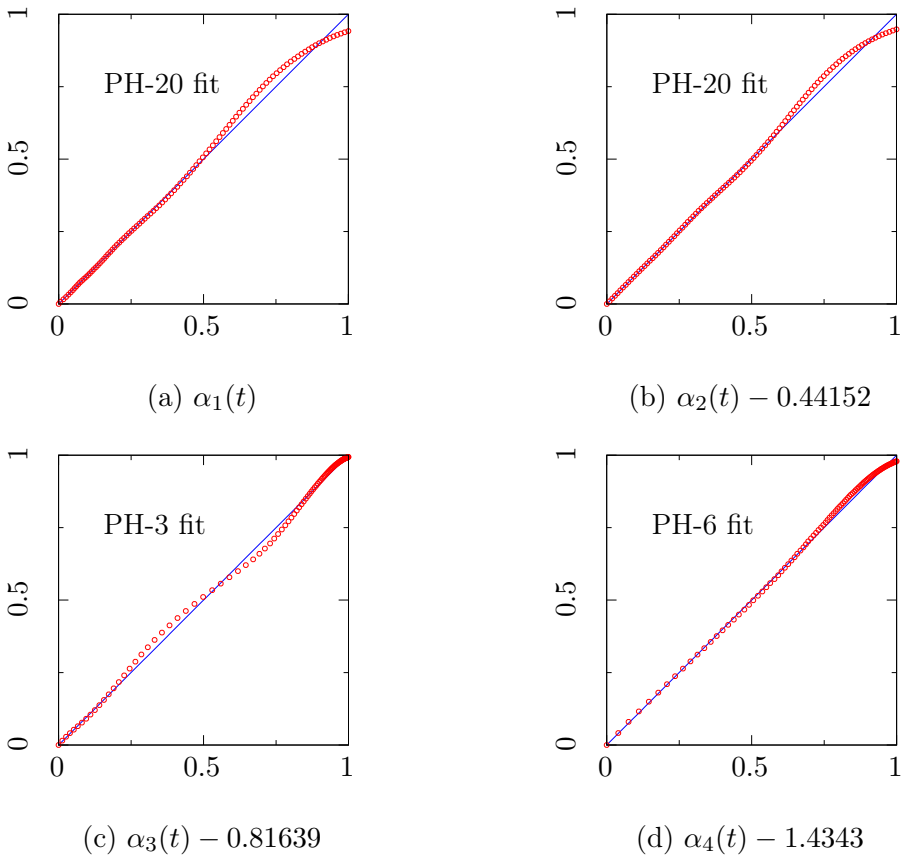
(a) τ_1-1 (b) τ_2-1

(c) τ_3-1 (d) τ_4-1

Figure 2.6: Probability plots of the phase-type fitting for sojourn times in each state

power generated by a panel of unit size is the solar irradiance effectively received multiplied by the efficiency of the panel) or a more detailed model such as the one implemented in the SolarStat tool [Gianfreda 2014]. The second option is to use directly a model for the power generated by a given PV panel (i.e., a model for i_{out}). Miozzo et al. have developed two such models in [Miozzo 2014]. In this section, we propose a modification to their on-off model. We will compare our modified model to theirs in Section 2.6 and also to the model of Section 2.4 after we translate the solar irradiance to generated power using the SolarStat tool. We present briefly the on-off model in [Miozzo 2014] before explaining our modification.

The dynamics of the harvested current $i_{\text{out}}(t)$ are captured by a two-state semi-Markov process. The distributions of the sojourn times and of $i_{\text{out}}(t)$ in each state are statistically defined using hourly measurements of the solar irradiance. In practice, Miozzo et al. apply the procedure summarized in Section 2.2.3 to map the solar irradiance data into the power generated by a PV panel of given size (number of solar cells connected in series/parallel) and characteristics (open circuit voltage, short circuit current, and reference temperature). Assuming the output voltage to be constant, the generated power and the output current are proportional to each other. The mapped data is grouped by month and for each month the values of the *output current* $i_{\text{out}}(t)$ are classified into two states according to an arbitrarily low threshold. All points falling below the threshold correspond to the "night" state and points falling above the threshold correspond to the "day" state. The authors of [Miozzo 2014]



(a) $\alpha_1(t)$ (b) $\alpha_2(t) - 0.44152$

(c) $\alpha_3(t) - 0.81639$ (d) $\alpha_4(t) - 1.4343$

Figure 2.7: Probability plots of the phase-type fitting for $\alpha(t)$ values in each state

use kernel-smoothing techniques to estimate the distributions of the durations and output current in each state for every month of the year. The model is as follows: when entering a state, a current and a duration are drawn from the corresponding distributions, then the source outputs the drawn current *constantly* for the drawn duration. At the end of the drawn duration, the source switches its state. In practice, the output current in the night state is set to 0.

Modified On-Off Model. To better capture the fluctuations observed in the solar irradiance $I_G(t)$ (which will inevitably be present in $i_{out}(t)$), we propose to modify the above-mentioned model in the following way: instead of keeping the current *constant* during the time the process remains in the "day" state, we frequently *resample* (every ten minutes) from the current distribution until a transition occurs.

2.6 Results

In this section, we will evaluate the models presented in Sections 2.4 and 2.5. We consider first the autocorrelation function (ACF) as a metric to test how well do generated synthetic data match the empirical data according to second order statistics. The autocorrelation function definition is presented in Appendix B. The empirical data is a 5-year long set of output

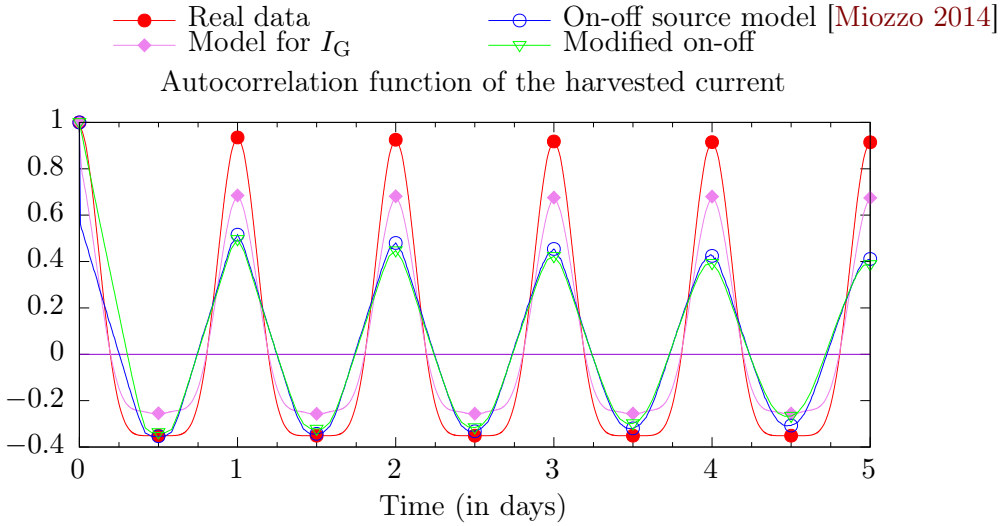


Figure 2.8: ACF of the current harvested using Panasonic solar panels

current values sampled every minute. The current values are those matched by SolarStat (for a Panasonic solar panel of unit size) for the solar irradiance measurements [Andreas 2012]. In case of simulations with per-minute data, we modify the SolarStat in order to run properly. We generate three synthetic data that are:

1. a 5-year long set of output current values sampled every minute using the model of the solar irradiance presented in Section 2.4 and SolarStat to translate the irradiance into output current;

2. a 5-year long set of output current values sampled every 10 minutes using the on-off model in [Miozzo 2014];

3. a 5-year long set of output current values sampled every 10 minutes using our modified on-off model (Section 2.5).

The autocorrelation functions of these four data sets are depicted in Fig. 2.8. Our solar irradiance model performs fairly well, capturing most of the correlations present in the empirical data. As already found by the authors of [Miozzo 2014], the ACF of the on-off source model poorly resembles that of the empirical data. The ACF of our modified on-off model performs seemingly equally badly.

Strong correlations in the solar power exists over yearly lags due to the earth's annual circumnavigation of the sun. To assess how well does our solar irradiance model capture the correlations over very long periods, we sample the ACFs every 30 days and display the values in Fig. 2.9. We can make three observations: first, the ACF of the real data confirms the expected strong annual correlation; second, our solar irradiance model exhibits correlations that mimic those in the real data, even though to a lesser extent; third, the on-off models fail to track the ACF of the real data.

To complete this comparative analysis of the models, we compute the root mean square error (RMSE) between the ACF of the empirical data set and that of each of the synthetic data set. The RMSE metric is as follows: $\text{RMSE} = \sqrt{\frac{1}{n}\sum_i^n (y_i - \hat{y}_i)^2}$, where y_i and $\hat{y}_i$ are the ith samples of the empirical and synthetic data respectively, and n is the number of samples.

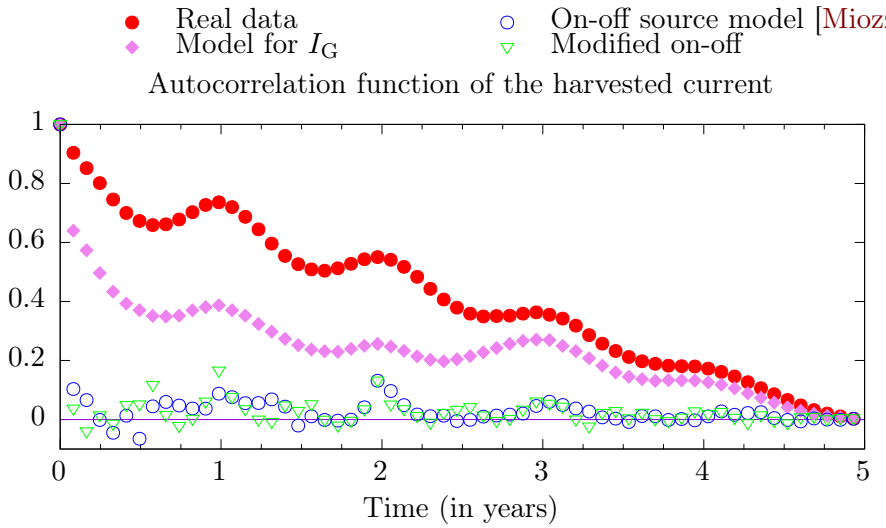


Figure 2.9: Samples of the autocorrelation function of the output current (one sample per 30 days)

Table 2.4: Root mean square error (RMSE) between real and synthetic data

Model of solar	Model of harvested power	
irradiance I_G	On-off source model [Miozzo 2014]	Modified on-off
0.1274	0.3231	0.2839

The results reported in Table 2.4 confirm the superiority of the solar irradiance model over the on-off models.

We can conclude from the comparison of the ACFs that our model of the solar irradiance outperforms the on-off models of the output current and captures well the multiscale correlations found in the real data.

We consider next the periodograms or power spectral density (PSD) of the empirical data set and the synthetic data set generated by the solar irradiance model (see Section 2.4). The spectral analysis allows to determine which characteristic time-scales are reproduced by the model. More details about periodogram are described in Appendix C.

We compute the periodogram using the function with the same name in the Signal Processing Toolbox of Matlab. We adjust appropriately the x-axis in order to have frequencies (f, in Hertz) instead of the angular frequency ω. The power spectrum densities are depicted in Figs. 2.10 and 2.11.

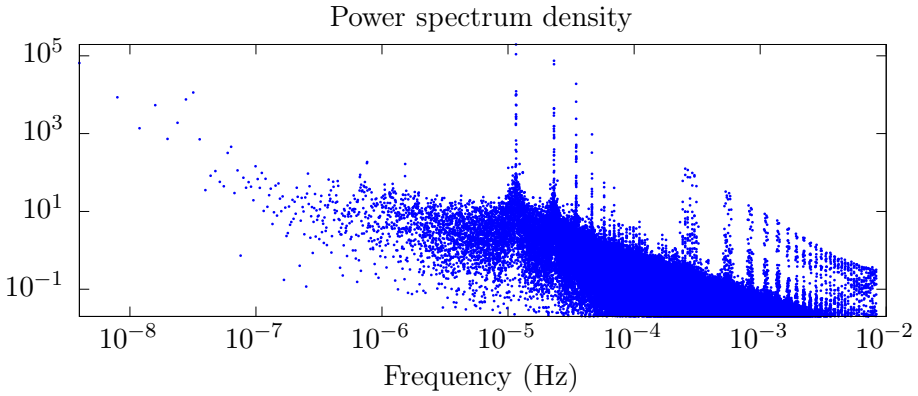


Figure 2.10: Power spectrum of the 1-minute values of output current mapped from the real measurements [Andreas 2012] by the SolarStat tool

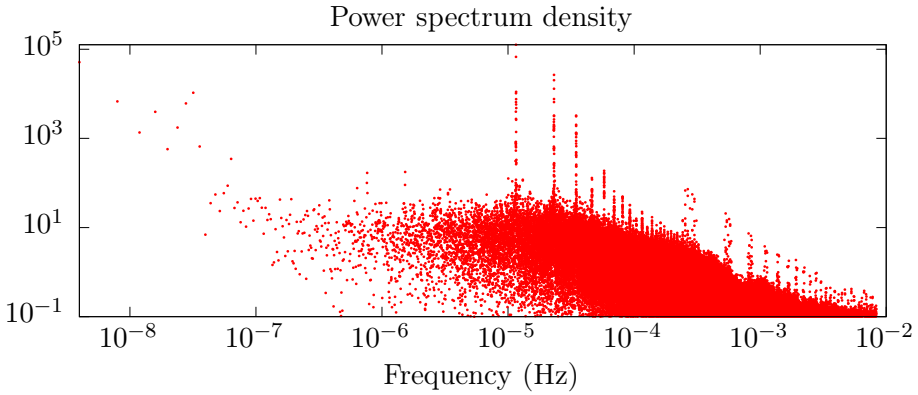


Figure 2.11: Power spectrum of the 1-minute values of output current obtained after generating a 5-year trajectory from the model of Section 2.4 and translating it to current with the SolarStat tool

Observe that Gu et al. have analyzed in [Gu 2001] the power spectrum of a 2-month set of 1-minute measurements of solar irradiance. The PSD had two clear peaks corresponding to 24 and 12 hours but other than those the absence of other characteristic time-scale was striking. This is not the case of the PSD of the real data set displayed in Fig. 2.10. We can observe a series of peaks at larger frequencies that are the harmonics of $1.157\overline{407}\ 10^{-5}$ Hz (which corresponds to 24 hours). The same observation applies to the PSD of the synthetic data set displayed in Fig. 2.11. The peak at the fundamental frequency corresponding to 1 day is clearly visible as well as those of its harmonics frequencies.

We conclude this section by stating that our solar irradiance model is able to generate synthetic data that exhibits all of the frequency peaks of real data, capturing its characteristic time-scales.

2.7 Discussion

We have developed in this work a stochastic model for the solar irradiance. The model combines a deterministic model of the clear sky irradiance with a stochastic model for the so-called clear sky index to obtain a stochastic model for the actual irradiance hitting the surface of the earth. As per-minute solar irradiance data is used to tune our model, we are able to

capture small time scales fluctuations as would be needed by ICT applications. Computing autocorrelation functions and periodograms of empirical and synthetic traces we found that our solar irradiance model performs very well. We believe our model can be used not only in the mathematical analysis of energy harvesting communication/computer systems but also in their simulation.

Chapter 3

Towards Modeling Real Data Center Clusters

Contents

In this chapter, we propose a model for computing clusters. In such systems, jobs to be executed are submitted by users. These jobs may generate a large number of tasks. Some tasks may be executed more than once while other may abandon before execution. We develop a multi-server queueing system with abandonments and resubmissions to model computing clusters. To capture the correlations observed in real workload submissions, we consider a Batch Markov Arrival Process. The service time is assumed to have a phase-type distribution.

3.1 Introduction

Private or public computing clusters are *de facto* solutions to run parallel applications. Such infrastructures comprise of multiple compute nodes offering different resources, e.g, CPU, memory and storage. A user interacts with a cluster by submitting a *job*, which consists of one or more parallel *tasks* to execute. Each task describes an amount of resource to allocate to run a given application. The *cluster scheduler* is then in charge of deploying the tasks to suitable computer nodes to execute the job.

Performance evaluation of clusters is an established approach to study their usage and forecast their behavior in different circumstances. For example, the resulting models are often considered to generate synthetic but theoretically realistic workloads to prototype and evaluate cluster schedulers [Goiri 2015, Hu 2008, Goiri 2012, Liu 2012b]. Accordingly, the quality of the decisions made from such experiments depends to a large extend on the quality and the accuracy of the cluster model.

Many different models have been suggested for clusters, most of them being based on the Poisson assumption for inter-arrival or service time distributions and single task arrival [Jain 1991, Harchol-Balter 2013, Guo 2014, Mary 2012]. However, according to the literature in computing systems [Amvrosiadis 2017, Adaptive Computing 2018, Capit 2005, Reiss 2011], jobs are submitted grouped by batch. Customers may also abandon the system or resubmit their jobs while a robust scheduler may automatically resubmit crashed tasks or kill low priority jobs in favor of high priority jobs. We consider fundamentally impactful for the job life-cycles and may, in such case reduce the representativity of existing models thus their benefits to forecast properly the job behavior.

Lucantoni [Lucantoni 1991] first has investigated many one-server queues with Batch Markovian Arrival Process (BMAP) input. However, multi-server queues with BMAP input need more investigation. In [Baba 1983], an algorithmic solution for delivering the steady state features of the M/PH/c queue with batch arrivals is proposed. Many models have been suggested for cloud computing the last decades. Guo et al [Guo 2014] propose M/M/m queueing system for cloud computing and optimize the average wait time, queue length and number of customer. Recent works are devoted to unreliable multi-server queues with BMAP input with Markovian flow of breakdowns [Kim 2017].

In the past, some works have focused on retrial queuing models with BMAP inputs. In [Breuer 2002b] the most general multi-server retrial queue system with BMAP input and phase-type distribution for service time is analyzed. Stability and instability conditions for this model are derived and the algorithm for computing the steady state distribution of the system is elaborated. In [Kim 2013], a retrial model with BMAP inputs and a finite buffer for impatient customers is proposed. Customers may either leave the system forever or go to orbit in case that they are not served immediately. In [Dudin 2013], an extension of the previous model is analyzed, including non-persistent customers who leave the system forever after trying unsuccessfully to be served. In [Kim 2008], a generalized BMAP/PH/N retrial queueing model is investigated with unreliable servers. Breakdowns arriving in a common MAP cause failure of one of the busy servers. After that the broken-down server should be repaired during the time interval having PH distribution. A customer whose service was interrupted can join the orbit or leaves the system forever. In [He 2000], the complicated system of BMAP/PH/N/N+K type with PH-retrials times is analyzed. He

solve the important problem of finding the stationary distribution existence condition and obtain ergodicity conditions of two BMAP/PH/s/s+K retrial queues with PH-retrial times and impatient customers. In [Kim 2010], the BMAP/PH/N retrial queue system is analyzed. Even though, the above state of the art works do not capture the features of the real data center clusters as presented in brief above.

In this chapter, we propose a new model for data center clusters that supports batch job submissions and takes into account both impatient and insistent customers' behaviour. The model is a multi-server queuing model with Batch Markovian Arrival Processes (BMAP) input, phase-type (PH-type) service time distribution with resubmissions and abandonment. We show that this model captures the job life-cycle of complex and real cluster systems like a typical Google cluster [Reiss 2011]. Finally, we discuss the research oriented but also the technical challenges to overcome to be able to evaluate the performance of real data centers.

The rest of this chapter is structured as follow. Section 3.2 discusses the characteristics of cluster systems. Section 3.3 details the stochastic model while Section 3.4 shows the evaluation analysis of our model.

3.2 Cluster Systems

We discuss here the core principles of clusters and cluster schedulers. We then propose a life-cycle for the jobs that fit today's systems. This analysis summarises a study of TORQUE [Staples 2006] and OAR [Capit 2005], two open-source schedulers that are used in production inside thousands of clusters.

Customers submit their jobs to a single submission queue that aims at being emptied by the cluster scheduler. We consider that the jobs are picked according to a *first come, first served* policy. While this scheduling policy is pretty common in production ready systems [Capit 2005, Staples 2006], it may not reflect the actual Google scheduler policy which is not disclosed. To start running a job, the scheduler removes it from the queue and starts to deploy some of its tasks among the compute nodes. Furthermore, because the probabilities of having at least one failure increases with the cluster and the job size, today's clusters support automatic job resubmissions.

Clusters usually consider that the tasks are independent, executable in parallel, in any order, and not necessarily simultaneously. Furthermore, clusters usually enable task to share a single physical machine called task-colocation.

Both features aim at increasing the cluster hosting capacity thus reducing the running cost per job.

Based on the literature, we propose in Figure 3.1 the life-cycle of a job/task inside the cluster. We consider a submitted job/task is in the `enqueued` state. The `scheduled` state depicts a job/task that starts being executed on the cluster. The `leave queue` state depicts a job/task that is dequeued before it could be `scheduled`. The `finished` state indicates the end of the execution on the cluster. The `leave cluster` state denotes a terminal state where the job/task is never again seen in the cluster. Last, the `waiting` state depicts a job/task that finished using the resources of the cluster but is about to go to the `enqueued` state. We observe three loops inside the life-cycle as jobs/tasks in the `finished`, the `leave queue` state or the `waiting` state can/will be resubmitted.

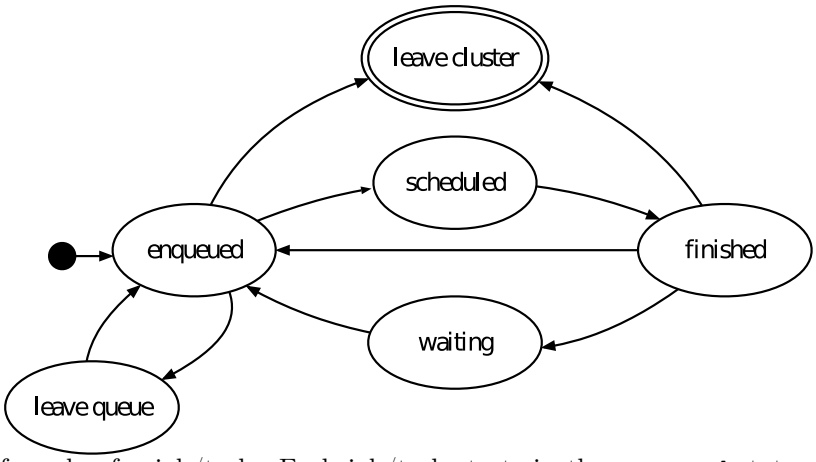


Figure 3.1: Life-cycle of a job/task. Each job/task starts in the `enqueued` state while the `leave cluster` is a terminal state.

Table 3.1 describes the possible *paths* for a job/task, i.e the state transitions of a job/task from the `enqueued` state to either a terminal state or a resubmission. The path 1 characterizes an impatient customer that removed its submission because it was useless or because the enqueued duration exceeded his expectations. The paths 2 and 3 characterize the most idiomatic transitions where a job/task is executed once or is resubmitted after a failure. Path 4 is a variation of path 3 resulting from the observation that finished jobs may wait significantly (more than 10 days) before being resubmitted.

Table 3.1: Possible paths for a job/task inside its life-cycle

Id	Path
1	`enqueued` → `leave cluster`
2	`enqueued` → `scheduled` → `finished` → `leave cluster`
3	`enqueued` → `scheduled` → `finished` → `enqueued`
4	`enqueued` → `scheduled` → `finished` → `waiting` → `enqueued`
5	`enqueued` → `leave queue` → `enqueued`

Paths 4 and 5 are the least likely of all paths. Therefore, our aim is to develop a queueing model at task level that captures the essential features of a cluster, considering that a task (or a customer in the queueing theory terminology) may follow one of the first three paths presented in Table 3.1.

3.3 Queueing Model

We model the cluster described in the previous section as a multi-server queueing system in which customers may abandon the queue while waiting for service. Upon service completion, customers may return to the queue. We denote the number of servers by c. There are multiple stochastic processes in this queue that we describe next.

Let $N(t) \in \mathbb{N}$, be the cumulated number of customers that has arrived to the queue until time t, and let $J(t) \in \mathcal{E}_A = \{1, 2, \ldots, A\}$, be the phase of the arrival process at time t. We assume that $\{(N(t), J(t)), t \geq 0\}$ is a Batch Markov Arrival Process (BMAP) [Lucantoni 1991]. The BMAP is a general arrival process that exhibits correlations over time and allows for cus-

tomers to arrive in batches. The BMAP is characterized by a set of matrices $D_k = [d_{ij}^{(k)}]_{i,j\in\mathcal{E}_A}$, for $k \geq 0$. For $k > 0$, $d_{ij}^{(k)} \geq 0$ represents the rate of transitions from phase i to phase j accompanied by the arrival of a batch of size k, while $d_{ij}^{(0)} \geq 0$ with $j \neq i$ represents the transition rate from phase i to phase j without any arrival, and $d_{ii}^{(0)} = -\sum_{j\neq i} d_{ij}^{(0)} - \sum_k \sum_j d_{ij}^{(k)}$.

Observe that $\{J(t), t \geq 0\}$ is a continuous-time Markov chain. Assume it has a unique stationary distribution $\boldsymbol{\theta}$. The customers arrival rate of the BMAP is then

$$\lambda = \boldsymbol{\theta}(\sum_{k=1}^{\infty} kD_k)\mathbf{1}. \tag{3.1}$$

Here $\mathbf{1}$ is a properly sized vector having all elements equal to 1.

Customers that are waiting for service are impatient. If the service of a customer does not initiate within a time that is exponentially distributed with parameter α, then this customer leaves the queue. The impatience durations are independent and identically distributed (iid).

Customers service times are iid random variables, furthermore independent of the arrival process and the impatience process. We assume they follow a phase-type (PH) distribution having S phases and representation $(\boldsymbol{\beta}, \boldsymbol{T})$. The cumulative distribution function of the service time is then $1 - \boldsymbol{\beta}\, e^{\boldsymbol{T}x}\mathbf{1}$.

Observe that $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_S)$ is the probability row vector giving the initial phase of the service time. The non-diagonal elements of the matrix $\boldsymbol{T} = [t_{ij}]_{i,j\in\mathcal{E}_S}$ with $\mathcal{E}_S = \{1, 2, \ldots, S\}$ are the transition rates between the phases. Service completion while in phase i occurs with rate $t_{i0} = -\sum_{j\in\mathcal{E}_S} t_{ij}$. Once its service completed, a customer may return to the queue for an additional service. This occurs with a constant probability p. The resulting BMAP/PH/c queue with impatient customers and resubmissions is illustrated in Figure 3.2.

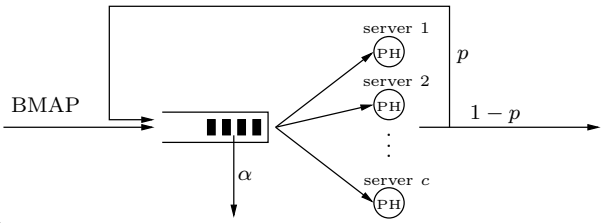


Figure 3.2: BMAP/PH/c queue with resubmission probability p and per customer abandon rate α.

The state of the system at time t is fully characterized by the triple $(X(t), \boldsymbol{S}(t), J(t))$. Here $X(t)$ denotes the number of customers waiting at time t and $\boldsymbol{S}(t)$ is an S-sized vector describing the state of the customers under service. To be more precise, the ith entry in $\boldsymbol{S}(t)$ is the number of customers whose service is currently in phase i. To ease the notation, we will use $|.|$ for the L_1 norm. Hence, the number of customers being served at time t is $|\boldsymbol{S}(t)|$ and those in the system amount to $X(t) + |\boldsymbol{S}(t)|$.

The most common approach to describe the service process at c servers, which work in parallel, having PH distribution is defined by the c-dimensional random process $h_t = \{h_t^{(1)}, h_t^{(2)}, ..., h_t^{(c)}\}$, $t \geq 0$, where $h_t^{(n)}$ is the phase of service process at the n-th busy server at time t. The dimension of the state space of the process h_t is S^c, where S is the number phases of PH distribution. In the case of data centers, the number of servers c is very large and the dimension of the state space of service process becomes even larger. This dimension of the

state space cannot fit at all in any computer memory and it is impossible to calculate the stationary distribution of the system. To overcome this problem, we decrease the dimension of the state space of the service process from S^c to $\binom{c+S}{S}$, describing the service process as $S = \{s_t^{(1)}, s_t^{(2)}, ..., s_t^{(M)}\}$, $t \leq 0$, where $s^{(q)}$ is the number of servers being in the q state of service time distribution. The dimension of S is $\frac{(s+M-1)!}{s!(M-1)!}$. Actually, as larger is the number of servers c, as larger is the difference between S^c and $\binom{c+S}{S}$.

It will be convenient to introduce the set $\mathcal{E}_m$ that contains all S-sized vectors $\boldsymbol{s} = (s_1, \ldots, s_S)$ such that $|\boldsymbol{s}| = m$ and $0 \leq s_\ell \leq m$. We define $\mathcal{E} = \cup_{m=0}^{c} \mathcal{E}_m$. Observe that the cardinal of $\mathcal{E}_m$ is $\binom{m+S-1}{S-1}$ and that of $\mathcal{E}$ is $\binom{c+S}{S}$.

Under the assumptions introduced in this section, the stochastic process $\{(X(t), \boldsymbol{S}(t), J(t)), t \geq 0\}$ is an $(S+2)$-dimensional continuous-time Markov chain over the state space $\mathcal{X} = (\{0\} \times \mathcal{E} \times \mathcal{E}_A) \cup (\mathbb{N} \times \mathcal{E}_c \times \mathcal{E}_A)$. One dimension represents the number of customers waiting in the queue, another one is for the arrival process and S is for the number of service phases. We enumerate the states of the process in lexicographic order of components $\{(X(t), \boldsymbol{S}(t), J(t))$.

We now proceed to the definition of the non-zero elements of its infinitesimal generator $\boldsymbol{Q}$, by reviewing all possible events.

The case when there is a change in the arrival process. There are four different situations in this case. In the first situation, there is a change in the phase of the arrival process without any batch arrival. In the second situation, actual batch arrivals occur, these differ according to whether the customers arriving in batch are all immediately handled by the corresponding number of servers. In the third situation, batch are partially handled such that some will have to wait. In the fourth situation, all new customers have to wait as all servers are busy. The elements of $\boldsymbol{Q}$ corresponding to these four situations are then

$$
\begin{aligned}
&q((n, \boldsymbol{s}, i), (n, \boldsymbol{s}, j)) = d_{ij}^{(0)}, \\
&\quad \text{for } i, j \in \mathcal{E}_A, j \neq i \\
&\quad \text{and for } n = 0,\ \boldsymbol{s} \in \mathcal{E},\ \text{or } n > 0,\ \boldsymbol{s} \in \mathcal{E}_c\ ; \\
&q((0, \boldsymbol{s}, i), (0, \boldsymbol{s} + \boldsymbol{r}, j)) = d_{ij}^{(k)} \binom{k}{r_1, \ldots, r_S} \prod_{\ell=1}^{S} \beta_\ell^{r_\ell}, \\
&\quad \text{with } \boldsymbol{r} \in \mathcal{E}_k, k > 0 \\
&\quad \text{for } \boldsymbol{s} \in \cup_{m=0}^{c-1} \mathcal{E}_m,\ \boldsymbol{s} + \boldsymbol{r} \in \cup_{m=1}^{c} \mathcal{E}_m,\ i, j \in \mathcal{E}_A\ ; \\
&q((0, \boldsymbol{s}, i), (k, \boldsymbol{s} + \boldsymbol{r}, j)) = d_{ij}^{(k+c-m)} \binom{c-m}{r_1, \ldots, r_S} \prod_{\ell=1}^{S} \beta_\ell^{r_\ell}, \\
&\quad \text{with } \boldsymbol{s} \in \mathcal{E}_m, \boldsymbol{s} + \boldsymbol{r} \in \mathcal{E}_c \\
&\quad \text{for } 1 \leq m < c,\ k \geq 1,\ i, j \in \mathcal{E}_A\ ; \\
&q((n, \boldsymbol{s}, i), (n + k, \boldsymbol{s}, j)) = d_{ij}^{(k)}, \\
&\quad \text{for } n \geq 0,\ \boldsymbol{s} \in \mathcal{E}_c,\ i, j \in \mathcal{E}_A\ .
\end{aligned}
$$

The case when there is a change in the service process. There are also four different situations in this case. In the first situation, the number of customers in the system is unchanged but some phases of those customers under service change. In the second and third

situations, one of the ongoing services ends causing the number of those waiting customers to decrease by one. However in the third situation, the distribution of the phases among the ongoing services remains the same. Observe that the third situation occurs also when a waiting customer abandons the queue. In the fourth situation, there are no waiting customers and an end of service turns one of the servers idle. To write the elements of $\boldsymbol{Q}$ corresponding to these four situations we will use the notation $\boldsymbol{e_i}$ to refer to a row vector (of dimension S) whose entries are null except the ith entry that is equal to 1. We can write:

$$
\begin{aligned}
&q((n,\boldsymbol{s},i),(n,\boldsymbol{s}+\boldsymbol{e_j}-\boldsymbol{e_\ell},i)) = s_\ell(t_{\ell j}+t_{\ell 0}p\beta_j),\\
&\qquad \text{for } i\in\mathcal{E}_A,\ j,\ell\in\mathcal{E}_S,\ell\neq j,\\
&\qquad \text{and for } n=0,\ \boldsymbol{s}\in\cup_{m=1}^{c}\mathcal{E}_m,\ \text{or } n>0,\ \boldsymbol{s}\in\mathcal{E}_c\ ;\\
&q((n,\boldsymbol{s},i),(n-1,\boldsymbol{s}+\boldsymbol{e_j}-\boldsymbol{e_\ell},i)) = s_\ell t_{\ell 0}(1-p)\beta_j,\\
&\qquad \text{for } n\geq 1,\ \boldsymbol{s}\in\mathcal{E}_c,\ i\in\mathcal{E}_A,\ j,\ell\in\mathcal{E}_S,\ell\neq j\ ;\\
&q((n,\boldsymbol{s},i),(n-1,\boldsymbol{s},i)) = n\alpha+\sum_{\ell=1}^{S}s_\ell t_{\ell 0}(1-p)\beta_\ell,\\
&\qquad \text{for } n\geq 1,\ \boldsymbol{s}\in\mathcal{E}_c,\ i\in\mathcal{E}_A\ ;\\
&q((0,\boldsymbol{s},i),(0,\boldsymbol{s}-\boldsymbol{e_\ell},i)) = s_\ell t_{\ell 0}(1-p),\\
&\qquad \text{for } \boldsymbol{s}\in\cup_{m=1}^{c}\mathcal{E}_m,\ i\in\mathcal{E}_A,\ \ell\in\mathcal{E}_S\ .
\end{aligned}
$$

The diagonal elements of $\boldsymbol{Q}$ are

$$
q((n,\boldsymbol{s},i),(n,\boldsymbol{s},i)) = -\sum_{(n',\boldsymbol{s}',i')\in\mathcal{X}} q\left((n,\boldsymbol{s},i),\left(n',\boldsymbol{s}',i'\right)\right)\ ,
$$

for $(n,\boldsymbol{s},i)\in\mathcal{X}$.

The infinitesimal generator, Q can be expressed as follows.

Lemma 1 *The infinitesimal generator Q of the Markov chain X_t, $t\geq 0$ has the following block structure:*

$$
Q=\begin{pmatrix}
Q_{00} & Q_{01} & Q_{02} & Q_{03} & Q_{04} & \cdots\\
Q_{10} & Q_{11} & Q_{12} & Q_{13} & Q_{14} & \cdots\\
0 & Q_{21} & Q_{22} & Q_{23} & Q_{24} & \ddots\\
0 & 0 & Q_{32} & Q_{33} & Q_{34} & \ddots\\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots
\end{pmatrix}.
$$

Before we analyse the sub-blocks, let us (re)define the following notations:
At time t:

- n_t be the number of customers in the waiting queue, $n_t\geq 0$
- A_t be the number of states of arrival phases, $A_t\geq 1$
- S be the number of states of PH-type distribution of service time, $S\geq 1$

- m_t be the total number of arrivals, $m_t \geq 0$

- $\vec{s}$ be the vector of service phase for service process.
  $\vec{s} = (s_1, s_2, ..., s_s) = \sum_{l=1}^{S} s_l \vec{e_l}, \vec{s} \leq 0, \forall l = 1, ..., s$, where $\vec{e_l}$ represents the standard basis for s-dimensional space.
  It represents how many active servers are in the respecting phase of service time distribution.

- p_j be the probability of starting a new service from j state

- K max size of a batch

For simplicity, we write n_t, A_t, m_t as n, A, m respectively in the following analysis. We define:

- $L = A * \binom{c+S-1}{S}$

- $L_0 = A * \binom{c+S}{S}$

In the sequel, we use the following notation:

- $\otimes$ is the symbol of the Kronecker product of matrices

- $diag(x)$, is a diagonal matrix with diagonal entries x, where x is a scalar or matrix

- $diag^j(x)$, is the element x in the j-th position of the diagonal of a matrix

- $diag_{-1}(x)$, is the first subdiagonal of a matrix with subdiagonal entries x, where x is a scalar or matrix

- I_j is an identity matrix of $j \times j$ dimension.

The sub-blocks $Q_{1,1+k}$ for $1 \leq k \leq K$ are $L \times L$ matrices and corresponds to the transition rates of arrivals of batch size k:

$$Q_{1,1+k} = \begin{cases} diag(D_k) \\ 0, \text{ otherwise} \end{cases}$$

where D_k contain the transition rates for which a batch size k occurs. Hence, for simplicity, $Q_{1,1+k}$ can be written as $Q_{1,1+k} = diag(D_k)$.

$Q_{n,n-1}$ are $L \times L$ matrices and corresponds to the end of service of one client and the start of service of another one. Its block structure is:

$$Q_{n,n-1} = \begin{cases} diag((n\,\alpha + \sum_{l=1}^{S} s_l\, t_{l0}\,(1-p)\,p_l) I_A) \\ (s_l\, t_{l0}\,(1-p)\,p_j) \times I_A, \text{ other blocks} \end{cases}$$

where I_A is an identity matrix of $A \times A$ dimension.

Let M be the $l \times l$ matrix with transition rates:

$$M = \begin{cases} diag(n\,\alpha + \sum\limits_{i=1}^{S} s_i\, t_{i0}\,(1-p)\,p_i) \\ s_i\, t_{i0}\,(1-p)\,p_j, \text{ otherwise} \end{cases}$$

then $Q_{n,n-1} = M \otimes I_A$, where $M = n\,\alpha\, I_c + (1-p)\,F$, where I_c is the identity matrix of $c \times c$ dimension (c is the number of servers as mentioned above).

Let F be the $l \times l$ matrix with transition rates:

$$F = \begin{cases} diag(\sum\limits_{i=1}^{S} s_i\, t_{i0}\, p_i) \\ s_i\, t_{i0}\, p_j, \text{ otherwise} \end{cases}$$

then $Q_{n,n-1} = n\,a\, I_{c+A} + (1-p)\,F \otimes I_A$, where $n \geq 2$ and I_A is the identity matrix of size $A \times A$ and I_{c+A} is the identity matrix of size $(A \times \binom{c+S-1}{S-1}) \times (A \times \binom{c+S-1}{S-1})$.

$Q_{1,0}$ is the $L \times L_0$ matrix and structure:

$$Q_{1,0} = \begin{pmatrix} 0 & 0 & \dots & 0 & Q_{n,n-1} \end{pmatrix},$$

where $n = 2$. Hence, $Q_{1,0}$ can be written as:

$$Q_{1,0} = \begin{pmatrix} 0 & \dots & 0 & 2\,a\, I_{c+A} + (1-p)\,F \otimes I_A \end{pmatrix}.$$

$Q_{n,n}$ is the $L \times L$ matrix which corresponds to the case that there is no departure or arrival but the servers may change service phase.

Let: $B_n(\vec{s}) = n\,\alpha + \sum_{l=1}^{S} s_l\, t_{l0}\,(1-p)\,p_l + \sum_{l=1}^{S} \sum_{\substack{j=0 \\ j \neq l}}^{S} s_l\,(t_{lj} + t_{l0}\,p_j)$ then,

$$Q_{n,n} = \begin{cases} D_0 - B_n(\vec{s})\, I_A, \ \vec{s} = \vec{s'} \\ s_l\,(t_{lj} + t_{l0}\,p\,p_j)\, I_A, \ \vec{s'} = \vec{s} + \vec{e_j} - \vec{e_l} \\ 0, \text{ otherwise.} \end{cases}$$

Let U_n be the square matrix of size ℓ with structure:

$$U_n = \begin{cases} -B_n(\vec{s})\,(N\,\vec{e_s}), \ \vec{s} = \vec{s'} \\ s_l\,(t_{lj} + t_{l0}\,p\,p_a)\,\mathbf{1}, \ \vec{s'} = \vec{s} + e_j \stackrel{\rightarrow}{-} \vec{e_l} \end{cases}$$

$Q_{n,n} = I_N \otimes D_0 + U_n \otimes I_A, n \geq 1$.

For the sub-blocks $Q_{0,0}$ and $Q_{0,k}$ the vectors $\vec{s}$ are ordered first according to $||\vec{s}|| = |\vec{s}|$,

then for the same $|\vec{s}| = m$, they are ordered lexicographically.

Each sub-block of $Q_{0,k}$ has cardinals $L_0 \times L$ and corresponds to the batch of customers' arrival of size k when the system is empty, for $1 \leq k \leq K$. Its block structure is:

$$Q_{0,k} = \begin{pmatrix} Q_{0k}^{(0)} \\ Q_{0k}^{(1)} \\ \vdots \\ Q_{1,1+k} \end{pmatrix}.$$

$Q_{0,k}^{(m)}$ represents the transitions from states $(0, \vec{s}, i)$ to $(k, \vec{s'}, j)$ with $|\vec{s}| = m$, $|\vec{s'}| = c$, $1 \leq i, j \leq$. $Q_{0,k}^{(m)}$ has cardinals $L \times A(\binom{m+S-1}{S-1})$ and its structure is:

$$Q_{0,k}^{(m)} = \begin{cases} diag(p_s^{c-m}\, D_{k+c-m}) \\ 0, \text{ otherwise.} \end{cases}$$

Let $P(m, m')$ be the stochastic matrix with the probabilities that $\vec{r} = \vec{s'} - \vec{s}$ has a given value $|\vec{r}| = |\vec{s'}| - |\vec{s}| = m' - m$, $0 \leq m < m' \leq c$. $P(m, m')$ block structure is:

$$P_{m,m'} = \left(\binom{m'-m}{r_1 \ldots r_s} \prod_{l=1}^{S} p_l^r \right)$$

then $Q_{0,k}^{(m)} = P(m, c) \otimes D_{k+c-m}$, $0 \leq m \leq D_{k+c-m}$.

Hence, $Q_{0,k}$ is:

$$Q_{0,k} = \begin{pmatrix} P(0,c) \otimes D_{k+c} \\ P(1,c) \otimes D_{k+c-1} \\ \vdots \\ P(c-1,c) \otimes D_{k+1} \\ I_c \otimes D_k \end{pmatrix}.$$

Q_{00} has cardinals $L_0 \times L_0$:

$$Q_{0,0} = \begin{pmatrix} Q_{00}^{(0,0)} & Q_{00}^{(0,1)} & \ldots & Q_{00}^{(0,c)} \\ Q_{00}^{(1,0)} & Q_{00}^{(1,1)} & \ldots & Q_{00}^{(1,c)} \\ 0 & \ddots & \ddots & \vdots \\ 0 & 0 & Q_{00}^{(c,c-1)} & Q_{00}^{(c,c)} \end{pmatrix}.$$

Let R_m be the following sub-matrix with $(\binom{m+S-1}{S-1} \times \binom{m-1+S-1}{S-1})$:

$$R_m = \begin{pmatrix} R_m^{(0,0)} & R_m^{(0,1)} & \ldots & R_m^{(0,m-1)} \\ R_m^{(1,0)} & R_m^{(1,1)} & \ldots & R_m^{(1,m-1)} \\ \vdots & \vdots & \vdots & \vdots \\ R_m^{(m,0)} & R_m^{(m,1)} & \ldots & R_m^{(m,m-1)} \end{pmatrix}.$$

Each sub-block of $R_m^{(i,j)}$ has cardinals $(\binom{m-i+S-1}{S-1} \times \binom{m-1-j+S-1}{S-1})$ and structure:

- For $j = i - 1$, $R_m^{(i,j)} = diag(i\,ti0)$

- For $j = i$

$$R_m^{(i,i)} = \begin{cases} diag^{(j)}((m-j)\,t_{s0}) \\ diag_{-1}(i\,t_{s-1,0}) \end{cases}$$

where $m \leq i \leq 0$ and $m - 1 \leq j \leq 0$.

$Q_{00}^{(m,m+k)}$ has cardinals $A \times (\binom{m+S-1}{S-1} \times \binom{m+k+S-1}{S-1})$ and structure:
$Q_{00}^{(m,m+k)} = P(m, m+k) \otimes D_k$, where $0 \leq m \leq c-1$, $1 \leq k \leq c$, $0 \leq m < m+k \leq c$

$Q_{00}^{(m,m-1)}$ has cardinals $A \times (\binom{m+S-1}{S-1} \times \binom{m-1+S-1}{S-1})$ and structure:

$Q_{00}^{(m,m-1)} = (1-p)\,R_m \otimes I_A$, where $1 \leq m \leq c$.
Let $B_0(\vec{s}) = \sum_{l=1}^{S} s_l\,t_{l0}\,(1-p) + \sum_{l=1}^{S} \sum_{\substack{j=0 \\ j \neq l}}^{S} s_l\,(t_{lj} + t_{l0}\,p\,p_j)$

$Q_{00}^{(m,m)}$ transitions from states $(0, \vec{s}, i) \to (0, \vec{s}, j)$ with $|\vec{s}| = |\vec{s'}| = m$, $1 \leq i, j \leq A$, where $\vec{s} = m$, we have $\binom{m+S-1}{S-1}$ possible vectors $\vec{s}$, they are ordered lexicographical:
$m\,\vec{e_s}, \vec{e_{s-1}} + (m-1)\,\vec{e_s}, 2\,\vec{e_s} + (m-2)\,\vec{e_s}, ..., (m-1)\,\vec{e_1} + \vec{e_2}, m\,\vec{e_1}$.

Each block of $Q_{00}^{(m,m)}$ has cardinals $A \times (\binom{m+S-1}{S-1} \times \binom{m+S-1}{S-1})$ and structure:

$$Q_{00}^{(m,m)} = \begin{cases} D_0 - B_0(\vec{s})\,I_A, \ \vec{s} = \vec{s'} \\ s_l(t_{lj} + t_{l0}\,p\,p_j)I_A, \ \vec{s'} = \vec{s} + e_j \vec{} - \vec{e_l} \\ 0, \text{ for other } \vec{s'}. \end{cases}$$

Let $U(m)$ be the stochastic matrix with cardinals $\mathcal{E} \times \mathcal{E}$ and transitions:

$$U_m = \begin{pmatrix} -B_0(m\vec{e_s}) & m(t_{s,s-1} + t_{s0}pp_{s-1}) & 0 & \dots & 0 & \\ t_{s-1,s} + t_{s-1,0}pp_s & -B_0(\vec{e_{s-1}}) + (m-1)\vec{e_s} & (m-1(t_{s,s-1} + t_{s,0}pp_{s-1}) & 0 & \dots & 0 \\ 0 & 2(t_{s-1,s} + t_{s-1,0}pp_s) & -B_0(2\vec{e_{s-1}} + (m-2)\vec{e_s}) & 0 & \dots & 0 \\ \vdots & \dots & \dots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & \dots & -Bo((m-1)\vec{e_1}) & t_{21} + t_{20}pp_1 \\ 0 & 0 & \dots & \dots & m(t_{12} + t_{10}pp_2) & -B_0(m\vec{e_1}) \end{pmatrix}$$

then:

- For $m = 0$, $Q_{0,0}^{(0,0)} = D_0$

- For $1 \leq m \leq c$, $Q_{0,0}^{(m,m)} = I_m \otimes D_0 + U(m) \otimes I_A$, where I_m is the identity matrix size $\binom{m+S-1}{S-1} \times \binom{m+S-1}{S-1}$.

Hence, $Q_{0,0}$ can be written as:

$$Q_{0,0} = \begin{pmatrix} D_0 & P(0,1) \otimes D_1 & P(0,2) \otimes D_2 & \dots & P(0,c) \otimes D_c \\ (1-p)\,R(1) \otimes I_A & I_1 \otimes D_0 + U(1) \otimes I_A & P(1,c) \otimes D_1 & \dots & P(1,c) \otimes D_c \\ 0 & (1-p)\,R(2) \otimes I_A & I_2 \otimes D_0 + U(2) \otimes I_A & \ddots & P(2,c) \otimes D_c \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & (1-p)\,R(c) \otimes I_A & I_c \otimes D_k + U(c) \otimes I_A \end{pmatrix}.$$

3.4 Evaluation Analysis

3.4.1 Terminology

We first present below the system terminology:
Offered throughput: rate of arrival of individual customers to the system, denoted λ;
Accepted throughput: rate of customers admitted to the queue, denoted λ_{acc};
Lost throughput: rate of customers rejected from the queue due to its finite capacity, denoted λ_{loss};
Reneging throughput: rate of customers who leave the queue due to impatience, denoted λ_{ren};
Serving throughtput: rate of customers admitted to service, denoted λ_{ser}.

3.4.2 Flow Conservations

Taking into account the probability p of a job reentering the queue, but when it does, it actually switches places with a job that was waiting in the queue (if any), we must have the flow conservation equations:

$$\lambda = (1-p)\lambda_{ser} + \lambda_{loss} + \lambda_{ren} \tag{3.2}$$

$$\lambda + p\lambda_{ser} = \lambda_{acc} + \lambda_{loss} \tag{3.3}$$

$$\lambda_{acc} = \lambda_{ren} + \lambda_{ser}. \tag{3.4}$$

The first one is the global conservation of customers that arrive to the queue and must depart, one way or another. The other ones are flow balances at the entrance and the exit of the buffer. There are three equations for the four variables, λ_{ser}, λ_{acc}, λ_{loss} and λ_{ren}, but one of these three equations is redundant.

In the general situation, the values of λ_{loss} and λ_{ren} depend on the queue distribution and cannot be evaluated without further analysis. There are some cases however where these values are easy to get: a) In the case where there is no impatience, $\lambda_{ren} = 0$; b) In the case where $N = \infty$, $\lambda_{loss} = 0$. Given the large number of servers in our system, we can assume that the buffer capacity can be possible infinity and satisfies the second case (b). Thus, when both conditions hold, we have:

$$\lambda_{ser} = \lambda_{acc} = \frac{\lambda}{1-p} \tag{3.5}$$

3.4.3 Flow Computations

Given the stationary distribution $\pi(n, \vec{s}, a)$ flow formulas are as follows. When customers arrive in batches, it is needed to specify the behavior when the whole batch does not fit in the queue. Given a number of customers n in the waiting room with capacity N and a batch of size k, we define the number of accepted/rejected customers respectively for the two following options:

1. Reject the whole batch ("b-rej"):

$$N_{acc}^{b-rej} = k\mathbf{1}_{\{k \leq N-n\}} \quad N_{rej}^{b-acc} = max\{0, k-N+n\}$$

2. Accept the part of the batch that fits in the queue ("b-acc"):

$$N_{acc}^{b-acc} = min\{k, N-n\} \quad N_{rej}^{b-acc} = max\{0, k-N+n\}$$

Using these, we have the formulas:

$$\lambda_{acc}^{b-rej} = p\lambda_{ser} + \sum_{n=0}^{N-1} \sum_{\vec{s}} \sum_{a} \pi(n, \vec{s}, a) \sum_{k=1}^{(N-n)^K} k \sum_{b} (D_k)_{ab} \tag{3.6}$$

$$\lambda_{loss}^{b-rej} = \sum_{n=0}^{N} \sum_{\vec{s}} \sum_{a} \pi(n, \vec{s}, a) \sum_{k=N-n+1}^{K} k \sum_{b} (D_k)_{ab} \tag{3.7}$$

$$\lambda_{acc}^{b-acc} = p\lambda_{ser} + \sum_{n=0}^{N-1} \sum_{\vec{s}} \sum_{a} \pi(n, \vec{s}, a) \sum_{k=1}^{K} min\{k, N-n\} \sum_{b} (D_k)_{ab} \tag{3.8}$$

$$\lambda_{loss}^{b-acc} = \sum_{n=0}^{N} \sum_{\vec{s}} \sum_{a} \pi(n, \vec{s}, a) \sum_{k=N-n+1}^{K} max\{0, k-N+n\} \sum_{b} (D_k)_{ab} \tag{3.9}$$

$$\lambda_{ren} = \sum_{n=0}^{N} \sum_{\vec{s}} \sum_{a} \pi(n, \vec{s}, a)(an) \tag{3.10}$$

$$\lambda_{ser} = \sum_{n=0}^{N} \sum_{\vec{s}} \sum_{a} \pi(n, \vec{s}, a)(\sum_{j} s^{(j)} T_{j0}) \tag{3.11}$$

When batches are of size one ($K = 1$), customers/batches are lost only when the queue is full ($n = N$), the rules "b-rej" and "b-acc" coincide, and these formulas simplify as:

$$\lambda_{acc} = p\lambda_{ser} + \sum_{n=0}^{N-1} \sum_{s} \sum_{a} \pi(n, \vec{s}, a) \sum_{b} (D_1)_{ab} \tag{3.12}$$

$$\lambda_{loss} = \sum_{s} \sum_{a} \pi(N, \vec{s}, a) \sum_{b} (D_1)_{ab} \tag{3.13}$$

When arrival processes are Poisson with rate λ, these formulas further simplify into:

$$\lambda_{acc} = p\lambda_{ser} + \lambda \sum_{n=0}^{N-1} \sum_{\vec{s}} \sum_{a} \pi(n, \vec{s}, a) \tag{3.14}$$

$$\lambda_{ren} = \sum_{n=0}^{N} \sum_{\vec{s}} \sum_{a} \pi(n, \vec{s}, a)(an) \tag{3.15}$$

$$\lambda_{ser} = \sum_{n=0}^{N} \sum_{\vec{s}} \sum_{a} \pi(n, \vec{s}, a)(\sum_{j} s^{(j)} T_{j0}) \tag{3.16}$$

3.4.4 Performance Metrics

Given the stationarity distribution $\pi(n, \vec{s}, a)$, we define the performance metrics: the expected number of servers and the expected waiting time are defined below as:

- Expected number of servers:

$$\mathrm{E}[|\boldsymbol{S}(t)|] = \sum_{(n,\boldsymbol{s},i)\in\mathcal{X}} |\boldsymbol{s}|\, \pi_{(n,\boldsymbol{s},i)} \ , \tag{3.17}$$

- Expected waiting time:

$$\mathrm{E}[W] = \frac{1}{\lambda_a cc} \sum_{(n,\boldsymbol{s},i)\in\mathcal{X}} n\, \pi_{(n,\boldsymbol{s},i)} \ , \tag{3.18}$$

3.4.5 The Arrival Phase Marginal Distribution

The arrival phase marginal distribution is denoted by, $\pi_A(a)$, where A is the number of arrival phase

$$\pi_A(a) = \sum_{n=0}^{N} \sum_{\vec{s}} \pi(n, \vec{s}, a). \tag{3.19}$$

Effectively, $\pi_A(a)$ expresses the probability that the customers arrive in the system in α phase. For all α, $\alpha \in [1, A]$ where:

$$\sum_{a=1}^{A} \pi_A(a) = 1. \tag{3.20}$$

3.4.6 The Queue Size Marginal Distribution

The queue size marginal distribution $\pi_Q(q)$ is denoted by:

$$\begin{aligned} \pi_Q(q) &= \sum_{n=0}^{N} \sum_{\vec{s}}^{c} \sum_{a}^{A} q\pi(n, \vec{s}, a) \\ &= \sum_{n=0}^{N} \sum_{\vec{s}}^{c} \sum_{a}^{A} (n + |\vec{(}s)|)\pi(n, \vec{s}, a) \end{aligned} \tag{3.21}$$

where, n is the number of customer in the waiting queue, N is the buffer size and, $q = n + |\vec{s}|$ is the queue size, including both customers in the waiting queue and customers that are serving. Effectively, this quantity shows the probability that q customers are in the system.

For all q, $q \in [0, N + c]$ where c is the maximum number of servers in the system:

$$\sum_{q=0}^{Q} \pi_Q(q) = 1. \tag{3.22}$$

3.4.7 The Busy-Server Marginal Distribution

The busy-server marginal distribution is defined as:

$$\pi_S(|\vec{s}|) = \begin{cases} \sum\limits_{|\vec{s}|=0}^{c-1} \sum\limits_{a=1}^{A} \pi(0, \vec{s}, a) \\ \\ \sum\limits_{a=1}^{A} \pi(0, |\vec{s}|, a) + \sum\limits_{n=1}^{N} \sum\limits_{a=1}^{A} \pi(n, \vec{s}, a), \text{ for } |\vec{s}| = c. \end{cases} \tag{3.23}$$

Effectively, $\pi_S(|\vec{s}|)$ shows the probability that $|\vec{s}|$ servers are occupied in the system. For all $|\vec{s}|$, $|\vec{s}| \in [0, c]$ where c is the maximum number of servers in the system:

$$\sum_{|\vec{s}|=0}^{c} \pi_S(|\vec{s}|) = 1. \tag{3.24}$$

3.4.8 The distribution of the number of servers in that phase

The number of server marginal distribution in that phase is denoted by:

$$\mathbb{P}(|\vec{s}| = m) = \sum_{k=m}^{c} p(k) \binom{k}{m} \beta_j^m (1 - \beta_j)^{k-m} \tag{3.25}$$

where β_j is the initial probability in phase j, m is the number of busy servers, $p(k)$ is the stationary probability that k servers are busy. We compute the probability over $\{0, 1, ..., c\}$. These probabilities for the same service phase should add up to 1.

3.5 Discussion

Modeling data center clusters is an essential prior to management activities such as capacity expansion, optimal energy consumption, *etc.* We propose in this study a task life-cycle in data center clusters, which corresponds to real situation. Based on that, we suggest and analyze the $BMAP/PH/N$ queueing model with abandonment and resubmissions. In the following, we show that queueing models of reasonable complexity can be built to match the complexity of real situations. In order to proceed to the numerical analysis of Google and Nef data center clusters. Before that, it is essential to characterize their workload in Chapter 4.

Chapter 4

Workload Characterization

Contents

In this chapter, we characterize in detail the workload of two different data center clusters and extract the appropriate parameters for our model, which is presented in Chapter 3. We use two datasets consisting of job scheduler logs: the first one is the well-known Google cluster dataset, a published 29-days cluster usage dataset, collected in May 2011 and the second one is the Nef cluster dataset from the data center cluster of the research center of French National Institute for Computer Science and Automation (Inria) in Sophia Antipolis. The Nef cluster dataset is collected from 1st July 2016 until 30th June 2018.

4.1 Introduction

Workload characterization plays an important role in capacity planning provision for real data center clusters. Hence, characterizing the workload of a single server and predicting its performance have been widely studied in the last decades [Barford 1998, Downey 1999, Artis 1979]. Even though, the real data center clusters have a large number of servers. Last years, research works have focused on workload characterization in cloud computing. Mishra [Mishra 2010] propose a methodology for workload classifications and its application to the Google Cloud Backend. Characterizing Google cluster dataset [Reiss 2011], conclude that machines are fairly homogeneous and the percentage of killed jobs is high. Then, Liu [Liu 2012a] propose a method to calculate the CPU cycles and they observe that many CPU cycles are put into jobs and tasks that will be eventually killed or fail. have noted that the google workload [Reiss 2011] is heterogeneous and quite dynamic and underline the need of innovating resource scheduling policies in cloud computing. Sharma [Sharma 2011] models and synthesizes task placement constraints in Google cluster, proposing a new metric for including constraints to resource. Then, Sharma [Sharma 2011] reproduce task scheduling delay and resource utilization by benchmarks of Google compute clusters. Wang [Wang 2015] discusses the resizing of the service capacity of data centers to match the workload and proposes a model to capture both slow time-scale non-stationarities at a slow time-scale and stochastic variability at a fast time-scale. In [Amvrosiadis 2018], three different types of data cluster workload are characterized, emphasizing its diversity and its impact on the result results. However, none of the above works characterize the arrival and the service process nor do they define any potential features as a basis for workload prediction model.

As mentioned in Chapter 3, we believe that the exponential assumptions related to arrival and service processes do not hold for the real data center clusters. In this chapter, we are going to assess this intuition and find other ways to model the above processes. We assume that the arrival process can be described by a Batch Markovian Arrival Process (BMAP) and the service process by a phase-type distribution. In the following analysis, we are going to attempt to validate these assumptions. Moreover, through this analysis, we are going to validate whether the abandonment ratio and resubmission probability features exist and, if yes, we are going to quantify them. Also, in case there is another feature which has not been previously considered, it will also be taken into account. For this reason, in the following, we characterize two different data center cluster workload types.

4.2 General Analysis

In this section, we discuss separately the global analysis of Google and Nef traces. The first trace is used internally by the Google employees and services and the second one is used for scientific purposes mostly by Inria employees. Before that, we should first refer in brief to some preliminary points of Google and Nef traces. In both cases, a *job* is defined as a Linux program which requires resources from the cluster scheduler to be executed. Jobs are independent and have their unique identifier given by the cluster scheduler. A job consists of one or more *tasks*. Each task represents a Linux program and runs on a single machine. The tasks of one job can be distributed across cluster nodes. An overview of the main characteristics of both traces is presented in Table 4.1.

Table 4.1: Characteristics of the traces

Task event table	Google Cluster	Nef Cluster
Machines	12,585	136
Jobs	672,005	5,284,650
Tasks	25,424,732	5,440,822
Events	144,648,288	33,660,798
Users	933	231
Size	15.41 GB	2.4 GB

In upcoming sections, we present and discuss the results of our general analysis for both datasets.

4.2.1 Google Cluster

In this section, we focus on analyzing Google traces [Reiss 2011]. The entire task event table consists of 144,648,288 records as presented in Table 4.1, each record reports on a single event. We counted the number of unique tasks that ran on each machine during the logging period. We remark that there are 12583 ($\sim$ 0.0005 %)tasks and 3966 jobs ($\sim$ 0.59% of jobs) that have started before the logging. Even though the number of jobs is 672,005, there are only 183955 distinct job names and just 39730 distinct logical job names. An explanation could be that people either do not give names to their jobs or can keep the same name for a different program out of laziness. We depict the empirical cumulative distribution of the number of submissions per user in Figure 4.1.

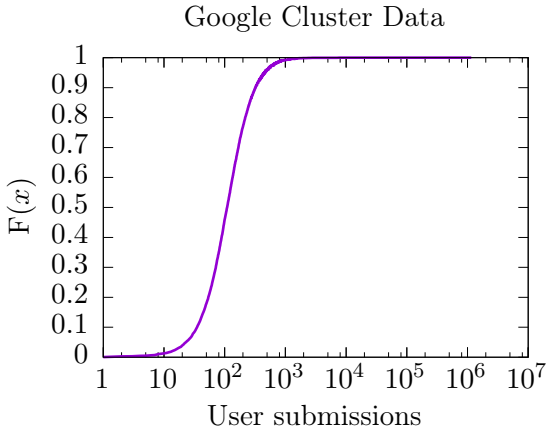


Figure 4.1: Empirical CDF of the number of submissions per user.

We process the traces in order to extract the history of each task from its first submission until its last recorded event. For each submission/resubmission, we obtain the following information:

- its job identifier and task index (within the job);
- if it is a resubmission after a fail/kill/lost event while in the queue, then the time elapsed from the event until the resubmission instant (this time is called “orbit”);

- if it is a resubmission after an end of execution, then the time elapsed from the end of execution until the resubmission instant (this time is called “resubmit”);
- the submission instant;
- the time spent in the queue waiting for service;
- the time spent in the service, if any;
- a flag recording the last event relative to the current (re)submission combined with an index indicating whether the task has been scheduled or not;

An example of the processing performed is illustrated in Table 4.2 and Table 4.3. Task 23 of job 5844213834 is resubmitted (line 2) after its service failed (line 1). It is subsequently scheduled (line 3), but the service fails again (line 4). Another resubmission (line 5) is scheduled (line 6) and no other event regarding this task is recorded in the task event table. This implies that the task was still under service when the logging ended. After processing, we obtain two records, one for each submission.

Table 4.2: Selected fields of Google task event table used in the processing

Selected fields of records of the task event table				
timestamp (μs)	jobID	task	machineID	event
⋮	⋮	⋮	⋮	⋮
1958394067572	5844213834	23	400468869	3 (event 3 = fail)
1958394067584	5844213834	23		0 (event 0 = submission)
1958425531958	5844213834	23	1094464	1 (event 1 = scheduling)
2403104028538	5844213834	23	1094464	3
2403104028550	5844213834	23		0
2403105165164	5844213834	23	351627471	1

Table 4.3: Selected fields of Google’s task event table used in the resulting information

Information obtained after the processing (all durations and timestamps are in microseconds)									
jobID	task	orbit	resubmit	submitTime	waitingTime	serviceTime			lastEvent
						complete	residual	age	jobScheduled
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
5844213834	23	12	1958394067584	31464374	444678496580				31
5844213834	23	12	2403104028550	1136614				103094834836	11

After processing the data at the level of tasks, we collect the following statistics reported in Table 4.4; the count of submissions, abandonments, scheduling, complete executions (and their nature), resubmissions and backlog at the end of the logging period.

Focusing on complete executions we compute the distribution among the possible end events, namely, “successful” (event 4 in the task event table), “evicted” (event 2), “failed” (event 3), “killed” (event 5), and "lost" (event 6). We observe that the Google Cluster trace had 7 missing end events. We also compute the ratio of resubmissions upon end of execution. Table 4.5 reports all these ratios.

Regarding resubmissions, we count separately those that occur within 30 μs after the end of the previous execution (these are referred to as “instant” resubmissions) and those that

Table 4.4: Statistics on tasks

Task event table	Google Cluster Data
Submissions	48,375,166
abandonments	1,000,020
still in queue	248
scheduling	47,374,898
Still in service	104,380
Complete executions	47,270,518
evicted (event 2)	5,864,353
failed (event 3)	13,829,769
successful (event 4)	18,217,975
killed (event 5)	9,350,102
lost (event 6)	8,312
missing end event	7
Resubmissions	22,929,812
instant (within 30μs)	22,481,988
delayed beyond 0.5 s	447,824

are delayed beyond 0.5 seconds. The reasons for this discriminative treatment becomes clear when looking at the resubmission delays displayed in Figure 4.2. There is a clear gap between very small values at the scale of micro-seconds and values larger than 0.5 seconds. We believe the resubmissions in both cases are not due to similar causes. This is supported by the figures reported in Table 4.16. According to Chapter 3, the life-cycle of a task of Google cluster is illustrated in Figure 4.3.

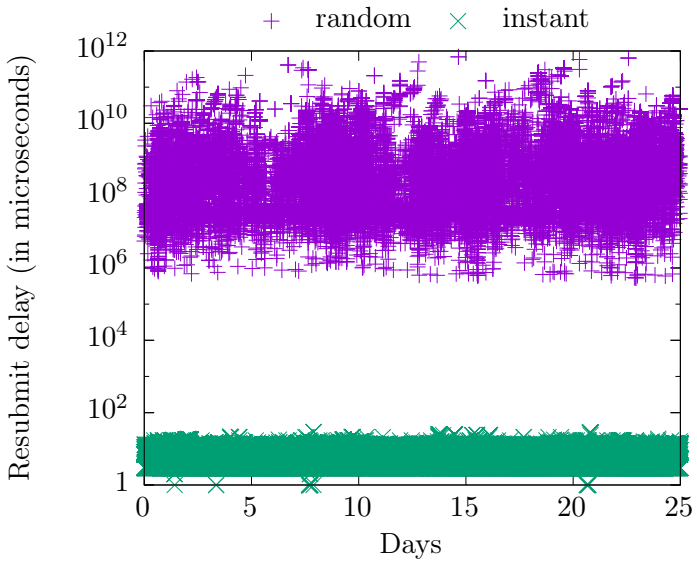


Figure 4.2: Delays observed before resubmissions during 29 days logging period.

We observe that there is no instant resubmission after a successful end of execution. Instant resubmissions occur mainly after unsuccessful terminations of an execution. By deriving the actual time spent in each part of the system (see columns 3–9 at the bottom of Table

Table 4.5: Statistics on complete executions of tasks

Task event table	Google Cluster data	Nef Cluster data
Type of end event		
evicted	12.41 %	-
failed	29.26 %	-
successful	38.54 %	84.51 %
killed	19.78 %	-
lost	0.02 %	7.39 %
missing end event	10^{-5} %	-
Resubmission upon completion		
instant (within 30 μs)	47.58 %	-
delayed beyond 0.5s	0.95 %	-
Total	48.53 %	2.58 %

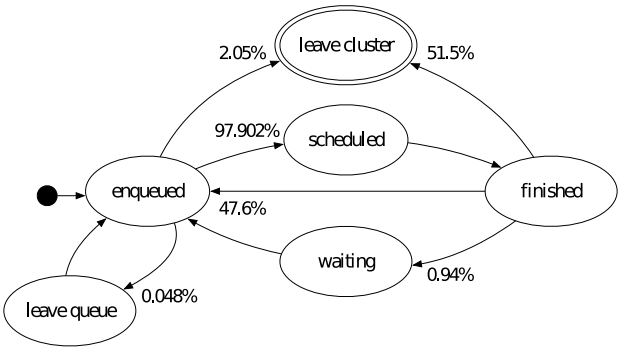


Figure 4.3: Google cluster data: Life-cycle of a task.

Table 4.6: Statistics on resubmissions according to the nature of the previous complete execution

Task event table		Google Cluster data
Nature of previous execution	Instant	Delayed
evicted	26.04 %	0.068 %
failed	61.13 %	0.016 %
successful	–	91.506 %
killed	12.81 %	8.409 %
lost	0.02 %	0.001 %

4.3), our processing allows us to characterize the different stochastic processes taking place in the system. We are now in position to proceed to the next step.

4.2.2 Nef Cluster

In this section, we introduce Nef cluster dataset consisting of job scheduler logs which are collected from the data center cluster of research center of the French National Institute for Computer Science and Automation (Inria) in Sophia Antipolis. The dataset covers 2 years, starting from 1st July 2016 untill 30th June 2018. Nef cluster is used for scientific purposes. Nef cluster uses OAR [d], an open source resource and task manager (or in other words

batch scheduler). In Nef traces, we have jobs and arrays or tasks and jobs respectively in the Google terminology. In the following, we use the terms jobs and tasks as defined in Google terminology. There are two execution types; terminated or so-called finished successfully and error. Error can occur either when a task leaves from the waiting queue before scheduling (abandonment) to the cluster or is lost. A lost task finishes and never terminates because of a problem when OAR was doing work internally. As reported in Table 4.5, the majority of tasks finish successfully (84.51%) and 7.39 % has been lost, the rest abandon. In case of resubmission, the scheduler gives to the job a new identifier but the information of old identifier is provided in the new submission.

The entire task table consists of 5,440,822 records as presented in Table 4.1, each record has one task identifier (taskId) which corresponds to some fields of task event table. Each record of task event table reports on a single event. For each submission/resubmission, we process the data in order to obtain the information which is mentioned in Section 4.2.1. An example is provided in Table 4.7. Task 1333 of job 6517286 (arrayId) has task id 6518618 and is resubmitted (line 5) with new task identifier 6528311. 2.58 % of tasks have been submitted for the first time and resubmitted during the log time window. The fact that the timestamps are in long time scale (seconds) does not permit us to check for instant resubmission or delayed. An error (line 14) occurred when the task 6518618 has been scheduled for first time and then resubmitted and finally finished successfully (line 22 and 23).

In this dataset, jobs can be in the default, besteffort or big queues. In cases of default and big queue, jobs wait until the scheduler can reserve the requested resources. Both queues give the same maximum walltime [1] to each job which is 30 days, but they differ in terms of priority. The big queue has higher priority than the default one. Moreover, big queue permits users to reserve almost 3 times more resources than default queue. However, a user can have at most 2 jobs running in the big queue at a given time. Thus, the big queue should be used only for jobs that need more than the maximum user resource of the default queue. For mode details, one can refer to Nef documentation [c]. Jobs in the besteffort queue run without resource reservation. If there is available resource on the cluster, they are allowed to run. A best effort job can be killed by the scheduler at any time because a default job wanted the used resource and resubmitted automatically when there are available resources. Besteffort jobs obviously are not a priority and the maximum running duration is 3 days. Best effort jobs enable users to can use more resources at a time and enhance efficient cluster usage. Hence, besteffort queue is appropriate for short jobs (several hours) that can easily be resubmitted. We present the statistics of Nef cluster data on the different queue types in Table 4.9.

In Nef cluster, two job types are allowed to be executed; the first one is the passive ones and the second one is the interactive ones. As far as the passive jobs are concerned, the user specifies a script at the submission time. This script will be executed on the first reserved node. In case of parallel operations, the user define parallel resources via the script. As for the interactive jobs , the user interacts with the command shell. Nef cluster statistics on the different job types are presented in Table 4.8.

[1] It is the difference between the task finishing time and the corresponding starting time.

Table 4.7: Selected fields of Nef task event table used in the processing

Selected fields of records of the Nef task table (all timestamps are in seconds)									
taskId	state	queueName	submissionTime	startTime	stopTime	taskType	resubmitTaskId	arrayId	arrayIndex
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
6518618	Error	besteffort	1526040896	1526044362	1526044362	PASSIVE	0	6517286	1333
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
6528311	Terminated	besteffort	1526045700	1526063338	1526063428	PASSIVE	**6518618**	6517286	1333

Fields of records of the Nef task event table of task 6518618/6528311				
taskStateLogId	taskId	jobState	dateStart	dateStop
37252706	6518618	Waiting	1526040896	1526044420
⋮	⋮	⋮	⋮	⋮
37269664	**6518618**	toLaunch	1526044420	1526044420
⋮	⋮	⋮	⋮	⋮
37269667	**6518618**	Launching	1526044420	1526045612
⋮	⋮	⋮	⋮	⋮
37270048	6518618	*Error*	1526045612	0
37270051	6528311	Waiting	1526045700	1526063374
⋮	⋮	⋮	⋮	⋮
37323253	6528311	toLaunch	1526063374	1526063375
⋮	⋮	⋮	⋮	⋮
37323257	6528311	Launching	1526063375	1526063376
⋮	⋮	⋮	⋮	⋮
37323278	6528311	Running	1526063376	1526063428
⋮	⋮	⋮	⋮	⋮
37323409	6528311	Finishing	1526063428	1526063431
⋮	⋮	⋮	⋮	⋮
37323425	6528311	*Terminated*	1526063431	0

Table 4.8: Statistics on job types

Nef cluster data	
job type	
passive	interactive
99.63 %	0.37 %

Table 4.9: Statistics on queue types

Nef cluster data		
queue name		
default	besteffort	big
73.216 %	26.78 %	0.004

4.3 Abandonments

A job/task can wait in the queue for some time but might leave the cluster before being scheduled. We refer to such events as abandonment. The time which a job/ task spends waiting in the queue for service before it leaves the cluster is termed as "impatient" time.

It is noteworthy that the observed impatient times are biased i.e. a sample can be observed only if it is smaller than the current waiting time. The probability that the impatient time, is smaller than the waiting time can be estimated by computing the ratio of the tasks that

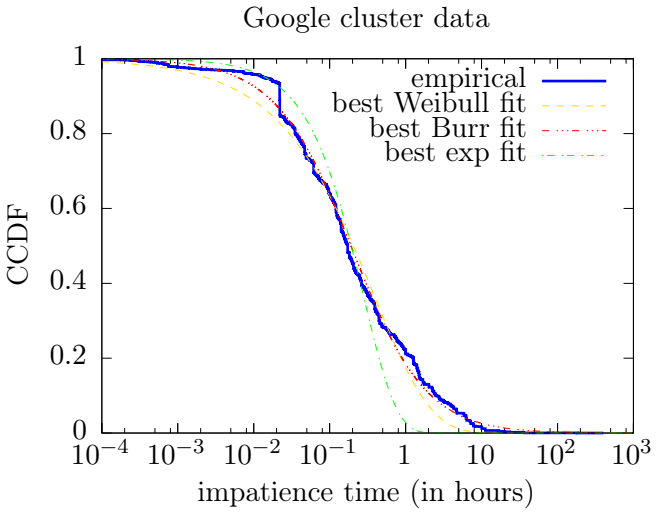


Figure 4.4: Complementary cumulative distribution function of the observed impatience times and best exponential/Weibull/Burr fit.

got abandoned before receiving service.

In the upcoming Section 4.3.1, we try to fit the Google and Nef impatience time distributions with the the following distributions: normal, Pareto, Nakagami, lognormal, inverse gaussian, Rayleight, exponential, Weibull and Burr. The fitting results of the last three ones are presented below. Details on Burr distribution are presented in Appendix D. Since the rest of the distributions do not agree fitting-wise with our empirical ones, they are not illustrated here. The fitting is obtained using the function `fitdist` [2] implemented in Matlab.

4.3.1 Traces

In Google traces, we have 979,398 samples of the time spent by a task waiting in queue for service before it leaves the Google cluster. The ratio of the tasks that got abandoned before receiving service is equal to 2.02 % in the Google cluster data. We illustrate the CCDF of the impatient times in Figure 4.4 and we depict the CCDF of the best exponential, Weibull and Burr fit.

In Nef traces, we have 440,450 samples of the time spent by a task waiting in queue for service before it leaves the Nef cluster. We illustrate the CCDF of the impatient times in Figure 4.5 and we depict the CCDF of the best exponential, Weibull and Burr fit. This ratio is equal to 0.0072 % in the Nef cluster data.

It is easy to observe, visually, in both cases that the best fitting corresponds to the Burr distribution.

4.4 Service Time

The service time is defined as the difference between the time a job/task is scheduled to the system and the time it is finished. Essentially, it is the time that system resources are occupied by a job/task.

[2] https://fr.mathworks.com/help/stats/fitdist.html

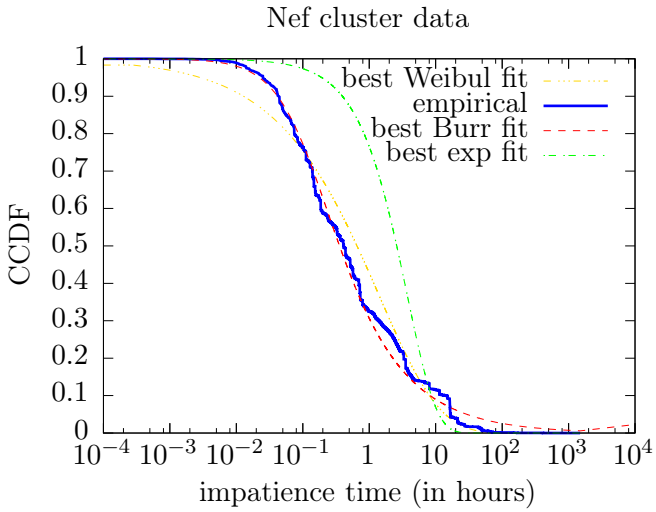
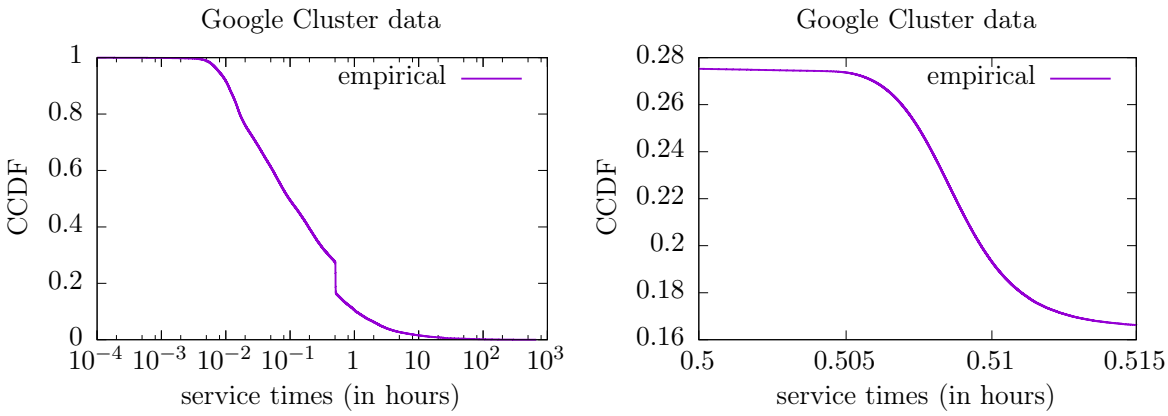


Figure 4.5: Complementary cumulative distribution function of the observed impatience times and best exponential/Weibull/Burr fit.

4.4.1 Google traces

In this section, we characterize the service times (at the level of tasks). The CCDF of the service times is depicted in Figure 4.6.



(a) Empirical & fitted distributions (b) Zoom around 0.5 hours

Figure 4.6: Google cluster data: CCDF of empirical and fitted distributions of the service times.

In this case, we collect all samples of service time from the Google trace [Reiss 2011]. These include executions that ended successfully or abnormally. Jobs and tasks in Google's cluster have a scheduling class ranging from 0 for non-production tasks like development to 3 for latency-sensitive tasks like those of revenue-generating user requests. In the following, we investigate whether a task's scheduling class affects its service time and the type of execution affects the CCDF of the service time.

For this, we exclude those executions with an unknown end event (either it is lost or it is missing). We observe that among all 47,149,520 executions with known end event, the largest portion (of size 15,263,643) consists of successful executions of class 0 tasks. Over all classes, successful executions constitute 38.62% of all executions. Across all types of executions, 79.27 % of them are of class 0 tasks.

Table 4.10: Average service time per scheduling class and for each type of execution (values expressed in hours)

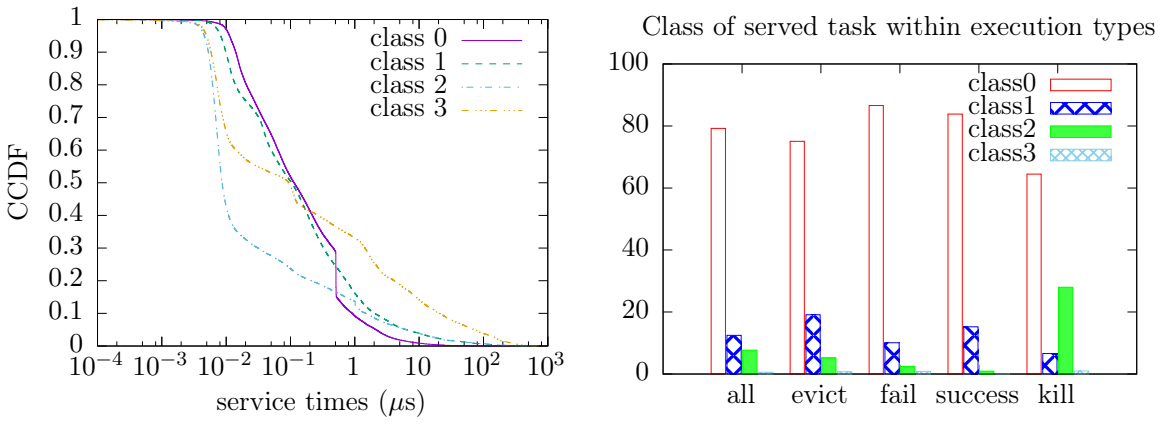
	Class 0	Class 1	Class 2	Class 3	Total
Evicted	1.896410	5.02017	6.83169	15.88630	2.83515
Failed	0.293587	1.13699	1.25747	0.81604	0.40668
Successful	0.451775	0.65381	0.53665	2.01595	0.48337
Killed	0.870284	10.05770	2.63013	26.88520	2.25293
All	0.635717	2.57573	2.75278	12.49110	1.10201

Table 4.11: Number of events across scheduling classes and execution types

	Class 0	Class 1	Class 2	Class 3	Total	
Evicted	4,372,337	1,112,145	292,973	38,580	5,816,035	12.34%
Failed	11,977,170	1,390,584	342,507	116,279	13,826,540	29.32%
Successful	15,263,643	2,775,974	168,388	60	18,208,065	38.62%
Killed	5,763,550	615,897	2,834,469	84,964	9,298,880	19.72%
Total	37,376,700	5,894,600	3,638,337	239,883	47,149,520	100%
	79.27%	12.50%	7.72%	0.51%	100%	

Aiming to assess whether a task's scheduling class affects its service time or not and whether executions of different types have different distribution, we build the CCDF of the service time for each execution type and each class. We also derive the CCDF within each class regardless of the execution type and for each execution type regardless of the class. We obtain in total 24 different distributions. We comment on these in the following sections.

4.4.1.1 Impact of the scheduling class on the service time



(a) CCDF of the service times within each class (b) Distribution of the class

Figure 4.7: Service time distribution for tasks within the same class (left) and percentage of service time samples within each execution type (right).

Figure 4.7a shows the CCDFs of the empirical distribution of the service time within each scheduling class. We observe no stochastic ordering between classes and service times. A

notable portion (about 10%) of class 0 service times corresponds to approximately 0.5 hour. This observation will be discussed later on.

We report in Figure 4.7b the distribution of the scheduling class within each execution type. We observe that samples of each execution type are mostly from class 0.

In Fig. 4.8 we derive the CCDF for a given execution type and a given class. Each graph display 5 curves illustrating the CCDFs for different scheduling class for the same execution type. Let σ_i^j be the service time of a class i task whose execution type is j and σ^j be a generic service time of execution type j. Here j takes value in {evict, fail, success, kill}. We can write:

$$P(\sigma_j > t) = \sum_{i=0}^{3} P(\sigma_i^j > t) P(\text{executed task is of class } i).$$

We observe in Fig. 4.10a that there is a stochastic ordering of the distribution according to the scheduling class. We have

$$\sigma_0^{evict} \leq_{st} \sigma_1^{evict} \leq_{st} \sigma_2^{evict} \leq_{st} \sigma_3^{evict}.$$

Evictions of lower classes occur stochastically earlier than those of higher classes. Fig. 4.8b shows a large density of values around 0.5 hour for class 0 which is due to 97.48 % of the failed executions of all 20,010 tasks of job 6336594489; these failed executions constitute 4,523,596 samples. This amounts to 9.59% of the total number of complete executions, hence these samples are also responsible for the density values observed in Figure 4.6b around 0.5 hours. A second large density of values is observed in Fig. 4.8b around 1 hour for class 2 tasks which is due to 92.84% of the failed executions of 4,860 tasks of job 6218406404; these constitute 53,804 samples. A plausible explanation is that the scheduler resubmits tasks that have failed but these fail over and over again with failures occurring roughly after the same execution time. This is not observed when tasks are evicted or when executions are killed.

In Fig. 4.8c, there does not seem to be much difference in the distribution among classes when the execution ends successfully. The only curve visually different does not serve as a good estimation of the distribution as there are only 60 samples of class 3 successful executions (see line 4 in Table 4.11). One would expect to see a stochastic ordering in Fig. 4.8d similar to the one observed in Fig. 4.8a. It is the case for classes 0, 1 and 3, but not for class 2. It seems that class 2 tasks are the ones killed preferably. However, the "kill" event is used in many different situations: cancellation by a user or by a driver, a failure, or when the cause of a task's termination is not precisely determined (see [Reiss 2011]).

4.4.1.2 Impact of the execution type on the service time

We display in Figure 4.9a the CCDFs of the empirical service time distribution for tasks having the same execution type. We observe larger service times for evicted executions. There is a strikingly large portion of about 0.5 hour service times in failed executions, this observation is consistent with the earlier discussion.

In Fig. 4.10, we group the service time distribution for tasks according to the scheduling class. Four graphs are presented. In each of the graphs, we illustrate 5 curves of the CCDFs which corresponds to different execution types { all types, evict, fail, success, kill }. Let σ_i be the service time of a class i task. We write the CCDF for a given class as follows:

$$P(\sigma_i > t) = \sum_j P(\sigma_i^j > t) P(\text{execution is of type } j)$$

In the case of class 2 tasks, as 78 % of the executions are killed (see Figure 4.9b) the CCDF $P(\sigma_2 > t)$ is mainly impacted by the CCDF $P(\sigma_2^{kill} > t)$. We can see in Fig. 4.10c that the curves labelled "all types" and "kill" are close to each other.

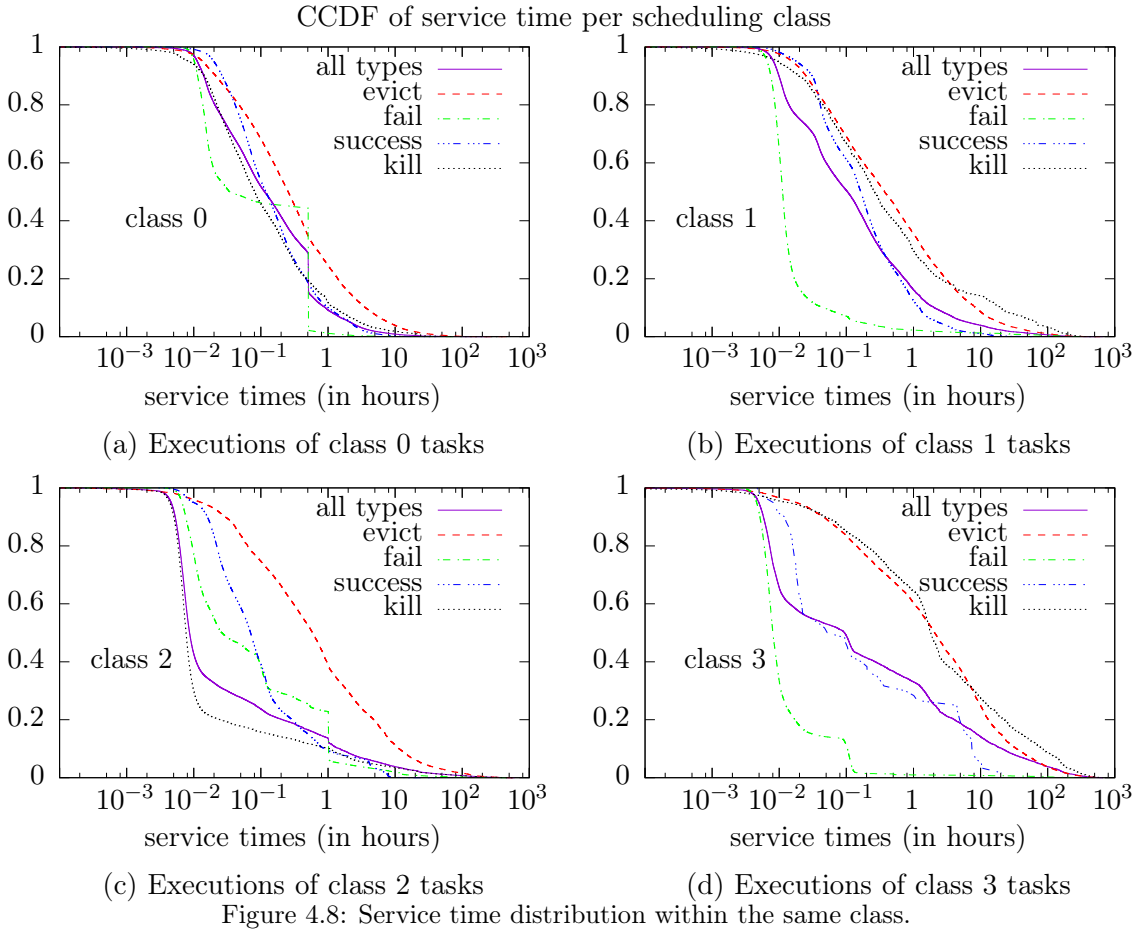
It is interesting to observe that, for any class, evictions occur on average later than the expected successful execution time. This observation made by looking at the curves labelled "evict" and "success" in Fig. 4.10 is confirmed by the average service times reported in Table 4.10. A successful execution is smaller on average than both an evicted execution and a killed execution, whatever the scheduling class. Failed executions are also smaller on average than evicted/killed executions.

4.4.2 Nef traces

In Nef cluster data, we have 5,000,364 samples of service time and we depict the empirical complementary cumulative distribution function in Figure 4.11.

4.4.2.1 Impact of the execution type on the service time

We display in Figure 4.11 the CCDFs of the empirical service time distribution for the tasks having the same execution type. We observe larger service times for lost executions. We can



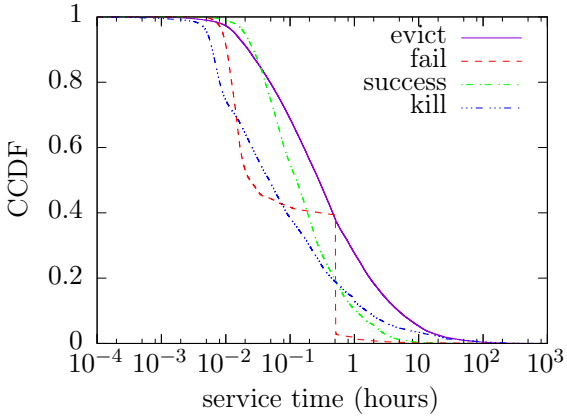
(a) Executions of class 0 tasks

(b) Executions of class 1 tasks

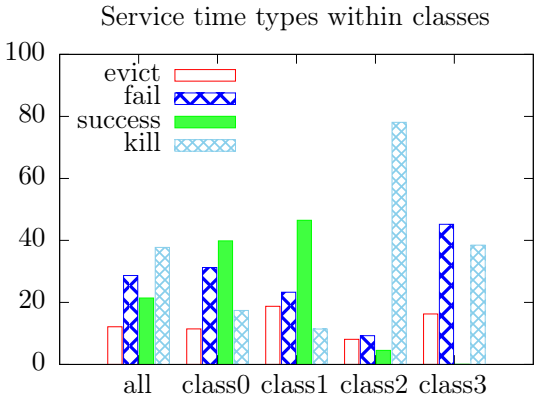
(c) Executions of class 2 tasks

(d) Executions of class 3 tasks

Figure 4.8: Service time distribution within the same class.

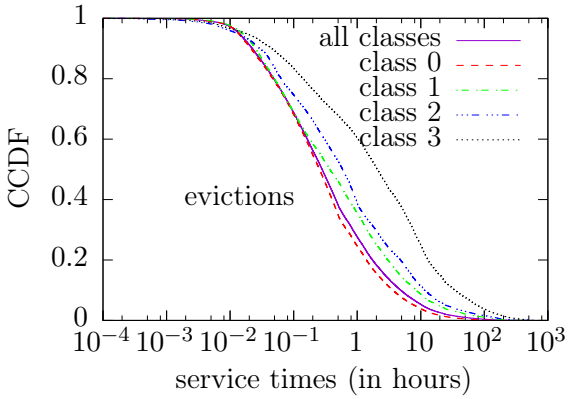


(a) CCDF of the service times per execution type

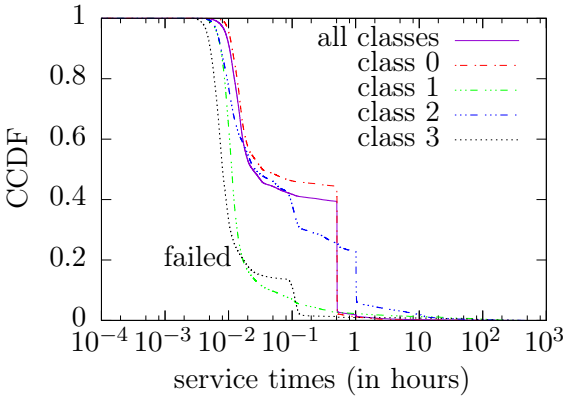


(b) Distribution of the event type

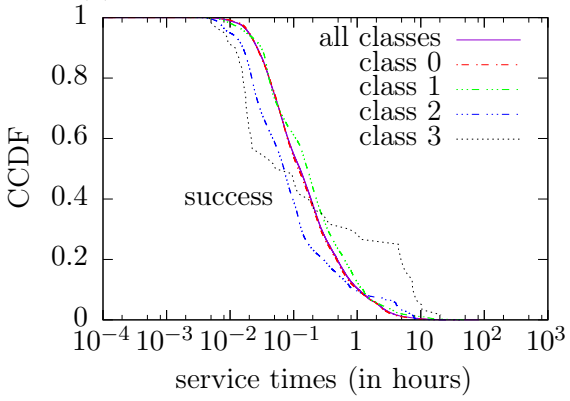
Figure 4.9: Service time distribution for tasks having the same execution type (left) and percentage of service time samples within each class (right).



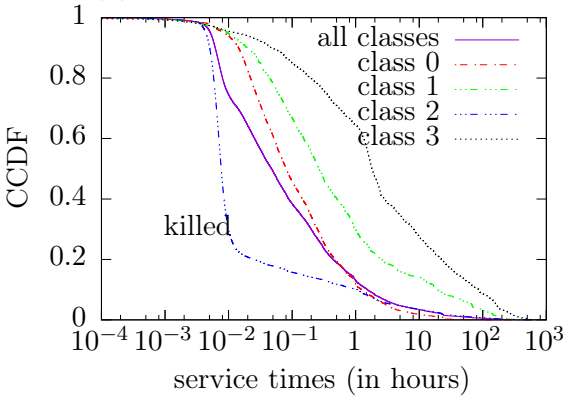
(a) Executions ending with an eviction



(b) Executions ending with a failure



(c) Executions ending successfully



(d) Killed executions

Figure 4.10: Service time distribution within the same execution type.

see in Figure 4.11 the the curves labelled "all types" and "success" are close to each other. Let σ_i^{nef} be the service time of a event type i task. We notice in Figure 4.11 that there is a stochastic ordering of the distribution according to the event types. We have

$$\sigma_{success}^{nef} \leq_{st} \sigma_{lost}^{nef} .$$

It is interesting to observe that, lost times occur on average later than the expected successful

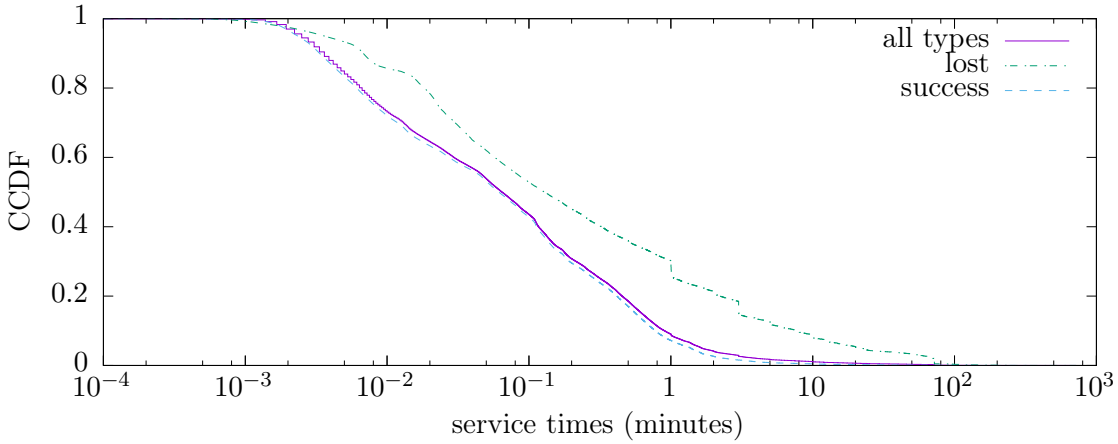


Figure 4.11: Nef cluster data: CCDF of service times per event type

execution time. This observation is confirmed by the average service times reported in Table 4.12.

Table 4.12: Nef cluster data: Average service time per event type

Event types	success	lost
	26.20 min	124.23 min

4.5 Arrival Time Characterization

Extracting the information about the arrival process requires additional processing. We need to identify the instant of an arrival (i.e. a submission) and the size of the batch that arrived at this instance.

4.5.1 Google

In the Google cluster data, the job event table records (among other things) the submission instants of jobs. Similarly, the task event table records the submission instants of tasks. There is a 2-microsecond gap between a job's submission instant and its first task's submission instant. Also, we observed the same gap between most of consecutive tasks' submission instants. Rarely, a gap larger than 2 μs is observed between the submission instants of consecutive tasks. Observe that only the first submission of each task must be considered, as resubmissions are accounted for through the resubmission probability. We use these information to identify the batch size of each arrival. We define a batch as a set of tasks that are submitted 2 microseconds apart. The batch arrival instant is the submission time of the first task in a batch. The range of batch sizes is from 1 to 90050 and the tasks arrival rate is 10.0885s^{-1} (see lines 2–3 of Table 4.17). We found also that even though most arrivals bring only one customer/task to the system, about 25% of the arrivals consist of more than one task and 5% consists of more than 100 tasks, the largest batch size observed being 90,050. We plot the probability mass function of the batch size in logarithmic scale in Fig. 4.12. Recall that the data trace contains 751 distinct batch sizes.

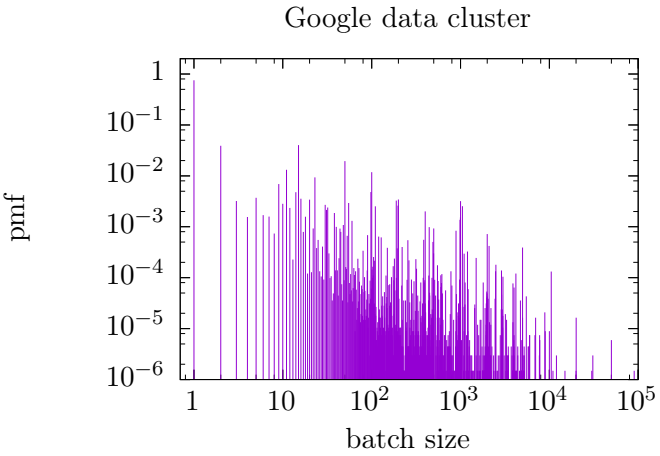


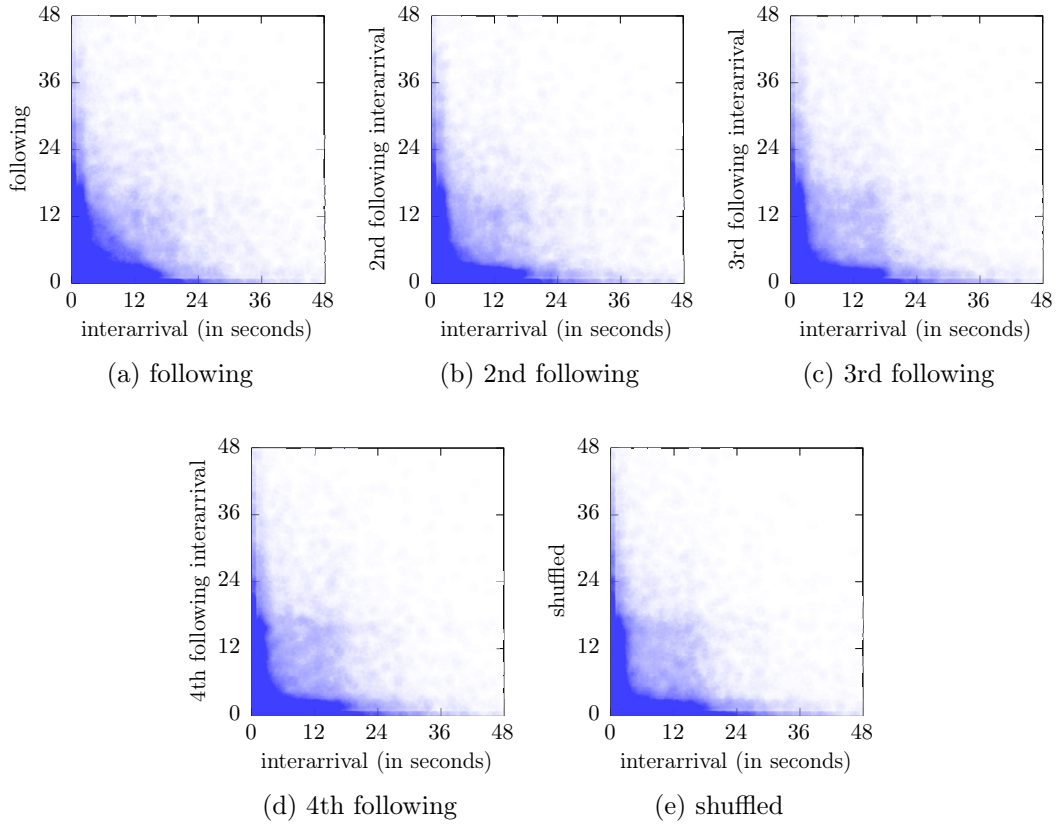
Figure 4.12: Probability mass function of the batch size for Google data cluster.

We then investigate whether there are correlations in the arrival process. We produce four scatter plots in which the interarrivals are on the x-axis and following ones are in the y-axis. For comparison, we shuffle the sequence of interarrivals and produce a scatter plot for the shuffled sequence. This scatter plot corresponds to no correlations in the sequence of interarrivals. All five scatter plots are presented in Figure 4.13. By comparing the scatter plots, we observe that Figure 4.13a is the one that differs the most from Figure 4.13e that has no correlations. On the other hand, the scatter plots in Figures 4.13d and 4.13e resemble each other. We can conclude that there are correlations between consecutive interarrivals and the correlations between pairs of interarrivals decrease as the interarrivals considered in each pair are farther apart.

4.5.2 Nef

In Nef data cluster, we define a batch as a set of tasks that are submitted at the same time (have same submit timestamp in seconds). Most of the times, a batch has the same array identifier (see Section 4.2.2). We have 533 distinct batch sizes. The range of batch sizes is from 1 to 10000 and the tasks arrival rate is $23.7015 s^{-1}$ (see line 2–3 of Table 4.29). We plot the probability mass function of the batch size in logarithmic scale in Figure 4.14. Observe that around 98% of jobs consist of one task.

Similarly, we investigate here whether there are correlations in the arrival process. We produce six scatter plots in which the interarrivals are on the x-axis and following ones are in the y-axis and compare them with the scatter plot for the shuffled sequence, as in Section 4.5.1. All six scatter plots are presented in Figure 4.15f. By comparing the scatter plots, we observe that Figure 4.15a is the one that differs the most from Figure 4.15f that has no correlations. On the other hand, the scatter plots with twentieth following in Figure 4.15e and 4.15f resemble each other. Observe that there are correlations between consecutive interarrivals and the correlations between pairs of interarrivals decrease gradually as the interarrivals considered in each pair are farther apart.



(a) following (b) 2nd following (c) 3rd following

(d) 4th following (e) shuffled

Figure 4.13: Google cluster data: Interarrivals are on the x-axis and on the y-axis we have (a) following interarrivals, (b) second following interarrivals, (c) third following interarrivals, (d) fourth following interarrivals, and (e) shuffled interarrivals.

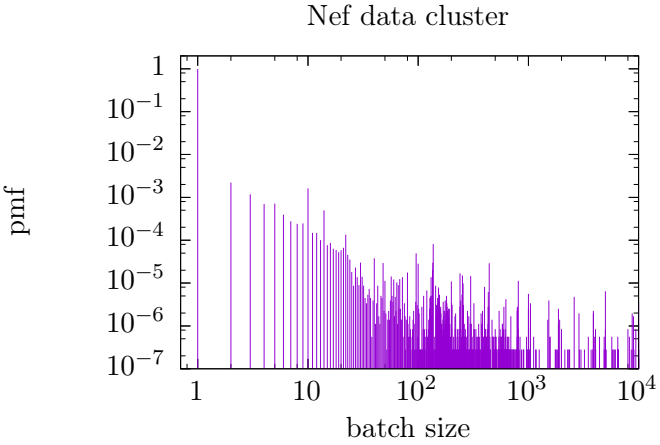
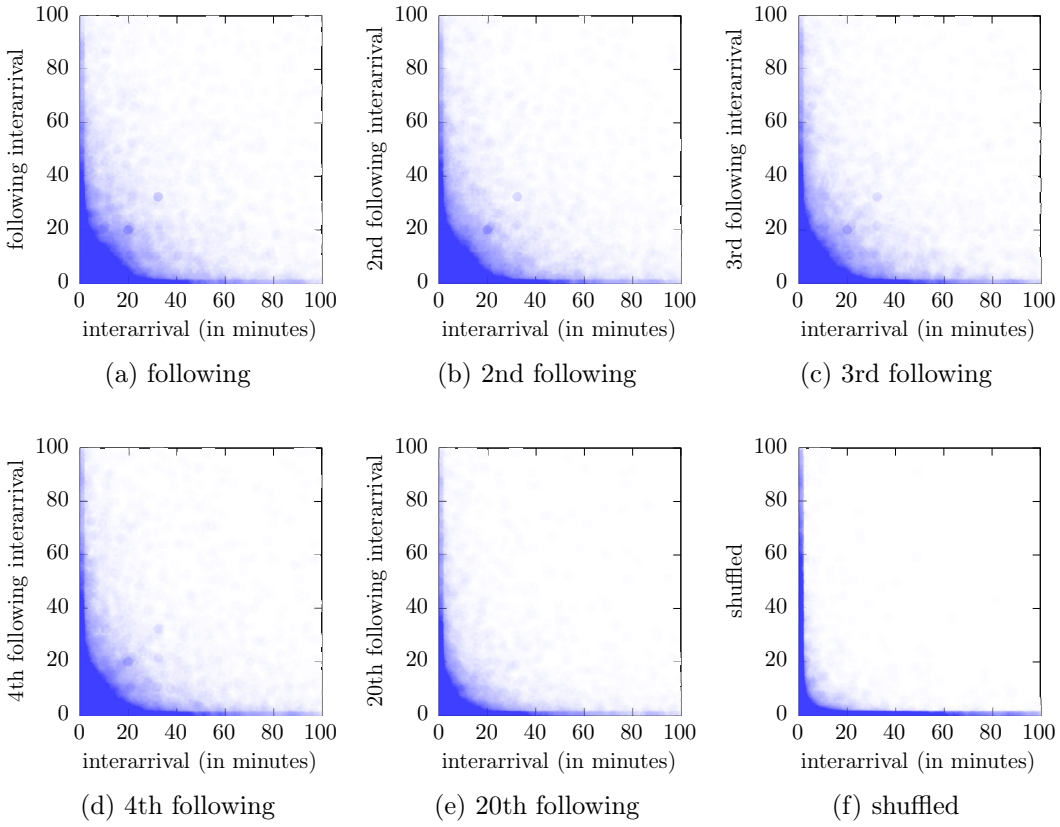


Figure 4.14: Probability mass function of the batch size for Nef Cluster

4.6 Waiting Time

The waiting time is defined as the difference between the time a job/task is submitted to the system and the time it is scheduled.

In Google traces, we have 48,218,459 samples of waiting time. We illustrate the complementary cumulative distribution functions (CCDF) of waiting times in Figure 4.16a. We



(a) following (b) 2nd following (c) 3rd following

(d) 4th following (e) 20th following (f) shuffled

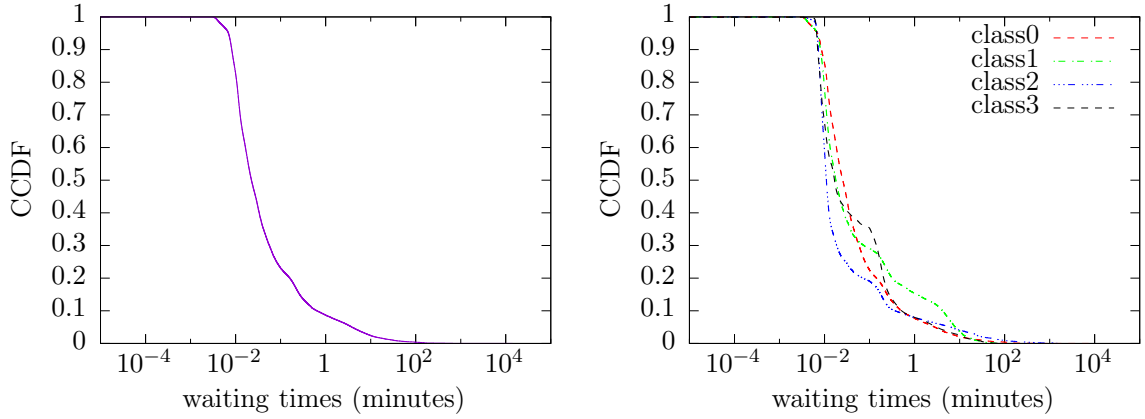
Figure 4.15: Nef cluster data: Interarrivals are on the x-axis and on the y-axis we have (a) following interarrivals, (b) second following interarrivals, (c) third following interarrivals, (d) fourth following interarrivals, (e) twentieth following interarrivals and (f) shuffled interarrivals.

then group the waiting times according to the scheduling class. In Figure 4.16 the CCDFs per scheduling class are illustrated. According to Google documentation [Reiss 2011], more latency-sensitive tasks (represented by class 3) usually have higher task priorities. Hence, it is expected that the waiting times of class 3, which represent the more latency-task, are less in comparison to those of the other three classes. However, as we can see in Figure 4.16 the waiting times of class 2 are most of the times the smallest ones. An explanation could be that the class 2 represents the second more latency-task and small waiting times are expected as well. Moreover, only 0.53 % of tasks belongs to class 3 and obviously not all tasks have high priority. Even though, among the average of waiting times per scheduling class, the class 3 has the smallest one, as reported in Table 4.13. The average waiting times of class 2 is the second one and very close to one of class 3, following the ones of class 1 and class 0, as it is expected.

Table 4.13: Average waiting times of Google cluster per scheduling class.

Average waiting times				
Google cluster	class 0	class 1	class 2	class 3
5.2 min	4.5 min	3.8 min	2.7 min	2.56 min

In Nef traces, we have 5,905,513 samples of waiting time of Nef cluster dataset. We depict



(a) CCDF of the waiting times (b) CCDF of the waiting times per scheduling class

Figure 4.16: Google data cluster: (a) CCDF of the waiting times and (b) CCDF of the waiting times per scheduling class.

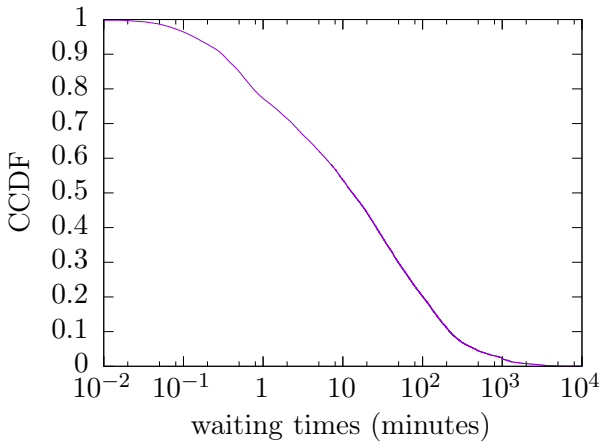


Figure 4.17: Nef data cluster: CCDF of waiting times

the CCDF of waiting times in Figure 4.17. The average waiting time of Nef cluster is 1.13 hour.

We conclude that the average time of Google cluster waiting time is much less than this one of Nef cluster.

4.7 Extraction of Model's Parameters from Traces

Obtaining the parameters of the model is a task of varying difficulty. The number of servers c is relatively easy to obtain from the specification of the cluster. The probability of a resubmission p can be estimated by counting the appropriate transitions in the life-cycle model of Section 3.2. It is estimated as the ratio of the number of instant resubmissions over the total number of executions. To isolate the per customer abandonment process, we identify those tasks that abandon the queue before receiving service and that are never resubmitted (the jobs to which they pertain leave the cluster). The waiting times of those tasks are samples of the per customer abandonment process. The impatience rate is estimated similarly. The identification of arrival and service characteristics is more complex.

4.7.1 Google traces

To overcome the technical difficulties faced when validating the queueing model against the Google cluster data, we extract two subtraces relative to a subset of the machines and use these for validation and comparison purposes. The characteristics of each subtrace are presented in Table 4.15. For each subcluster, we compute and present in Table 4.14 the statistics on the complete executions of tasks, we compare them to the ones of Google cluster, which are reported in Table 4.5. It is important to observe that the fate of a scheduled task does not vary much between the entire cluster and the two subclusters as the ratios reported in lines 3–6 of Tables 4.14 and Table 4.5 are comparable.

Resubmission probability. In the Google cluster, we observe that the resubmission probability is very large. Over 47 % of the tasks that end a service are resubmitted instantly (within 30 microseconds) for a new service. As we focus on a subset of 5 machines, it is expected to observe a much smaller resubmission probability as only instant resubmissions within the five machines of the subset are considered. Yet, there are still 1.6 % of instant resubmissions in subcluster-5b. The probability p is provided in line 5 of Table 4.17. Random and instant resubmissions are illustrated in Figure 4.19 for each subcluster.

Abandonment Process. We found 979,398 samples of tasks that abandon the queue before receiving service and that are never resubmitted. The time that elapses before a task abandons the queue is on average about 1.12 hours (see line 7 in Table 4.17). According to Section 4.3, there are three different distribution types that can fit the abandonment process; the Burr, the exponential and the Weibull ones, with Burr distribution to being the best one. However, in our model, for simplicity, we assume this time to be exponentially distributed. When the fitting criterion is to minimize the sum of squares of residuals, the exponential distribution that best fits the set of samples has the value 3.6 h^{-1} for the abandonment rate α. Observe that a fitting that matches the first moments yields $\alpha = 0.89\ h^{-1}$.

Service time. Having computed the service time for each task, it is easy to derive the average service time. Values are reported in line 4 of Table 4.17. The samples of the service time are used to characterize their distribution.

Number of servers. The number of servers c is not easily mapped to the specification of the cluster. The fact is that our queueing model assumes that a server (in the queueing terminology) can handle one and only one task at a time. This is not the case of the machines in the Google cluster. The machines table in the Google cluster data contains information about 12,583 machines. In the task event table, we can find 12,585 distinct machine identifiers. However, the number of tasks that are running simultaneously is on average 126,408 and ranges between 99,631 and 171,232. The same quantities for the two subclusters are listed in Table 4.18. Figure 4.18 depicts the actual number of tasks under service over time in the three cases considered. For completeness, we illustrate as well the number of tasks waiting in the queue and provide some measures on this quantity in Table 4.18. One possibility is to match the number of servers c with the maximum number of tasks under service. However this peak value corresponds to a system that would be

overdimensioned most of the time (see Figure 4.18) yielding a system utilization far from 1. It has been reported however that the scheduler of the Google cluster overcommits resources [[Amvrosiadis 2018],4.24] yielding a utilization close to 1. An alternative possibility is to select the number of servers c such that the system is stable. For each case, we use the arrival rate, the average service time and the resubmission probability to compute the load (see line 6 in Table 4.17). The number of servers c must be larger than the load.

Table 4.14: Subclusters: Statistics on complete executions of tasks

Task event table	Subcluster-5a	Subcluster-5b
Type of end event		
evicted	11.18 %	13.22 %
failed	29.26 %	30.36 %
successful	43.13 %	38.58 %
killed	28.50 %	17.80 %
lost	0%	0.04%
missing end event	0%	0%
Resubmission upon completion		
instant (within 30 μs)	5.17 %	1.67 %
delayed beyond 0.5s	1.62 %	1.39%

Table 4.15: Characteristics of the subtraces

Task event table	Subcluster-5a	Subcluster-5b
Machines	5	5
Jobs	2,964	3,968
Tasks	4,689	7,294
Events	15,084	22,664
Size	1.64 MB	2.47 MB

Table 4.16: Google subclusters: Statistics on resubmissions according to the nature of the previous complete execution

Task event table	Subcluster-5a		Subcluster-5b	
Nature of previous execution	Instant	Delayed	Instant	Delayed
evicted	95.753 %	11.18 %	30.4 %	13.46 %
failed	2.317 %	17.19 %	67.2 %	84.62 %
successful	–	38.58 %	–	0.96 %
killed	1.930 %	28.50 %	2.4 %	0.96 %
lost	–	–	–	–

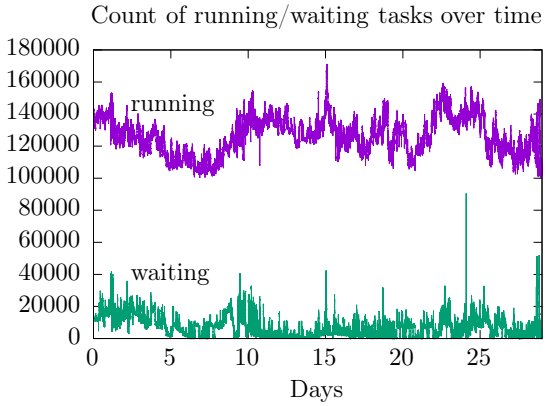
Table 4.17: Parameters extracted from the traces after processing

Task event table	Google Cluster data	Subcluster-5a	Subcluster-5b
Batch sizes	751 values in [1,90050]	1-5	1-6
Tasks arrival rate	10.0885 s^{-1}	5.8779 s^{-1}	8.9267 s^{-1}
Average service time	1.10201 min	2.05555 h	1.46790 h
Resubmission probability	0.4758	0.0517	0.0167
Load	76,345.36	12.74	13.11
Average impatience time	1.11980 h	n/a	n/a

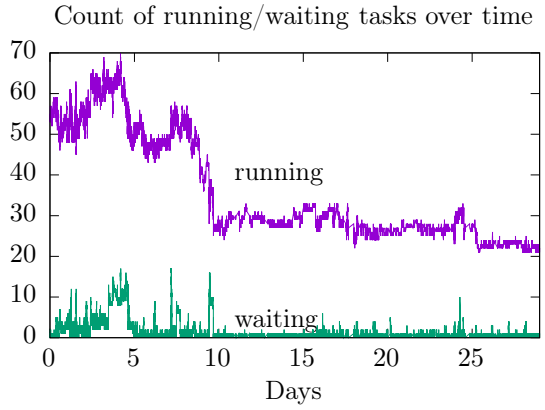
4.7.1.1 Fitting the service process of Google cluster data

We used the SPEM-FIT tool [3] to fit a PH distribution into the service times data trace. We have 47,157,789 samples of service time that we use in the fitting. The complementary cumulative distribution functions of empirical and fitted distributions are depicted in Figure 4.20. The CCDF of the fitted distribution with more than 5 states are visually non-distinguishable

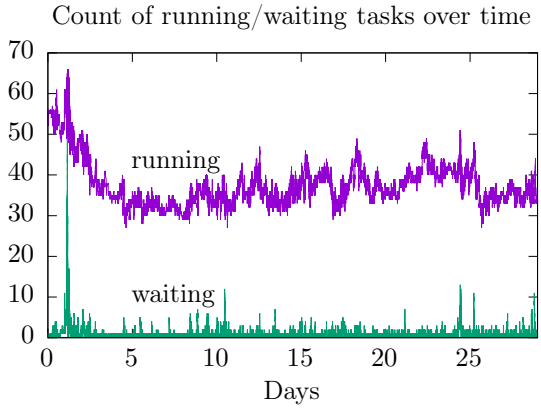
[3]The SPEM-FIT tool uses the Expectation-Maximization algorithm to estimate the parameters of the fitted PH distribution.



(a) Google cluster data



(b) Subcluster-5a



(c) Subcluster-5b

Figure 4.18: Evaluation of the number of tasks in the system during 29 days logging period.

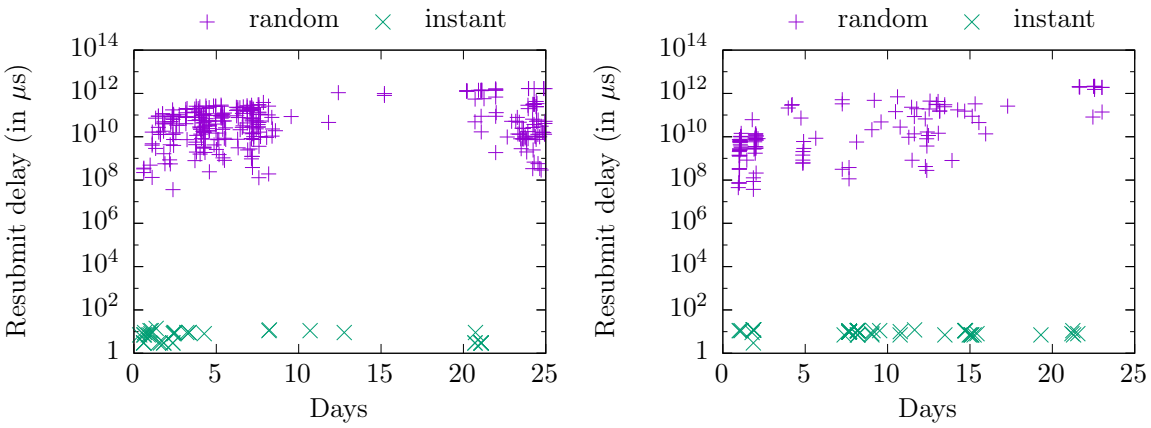
from the one with 5 states. A fitting with 5 states is needed if one seeks to match the tail of the distribution. Looking at the relevant metrics reported in Table 4.19, we can see that the quality of the fitting substantially improves when we increase the number of phases from 4 to 5 and then no substantial improvement is seen until the number of phases is 10. With a 10-phase distribution, the second moment of the fitted distribution approaches greatly the empirical second moment. According to the quality aimed, one can approximate the data with a 5-state or a 10-state PH distribution. If one seeks to minimize the complexity, then a 3-state PH distribution is a good tradeoff between complexity and fitting quality.

4.7.1.2 Subcluster-5a

We have 4,962 samples of service time that we use in the fitting. The CCDFs of empirical and some of the fitted distributions are depicted in Figure 4.21a. The CCDF of the fitted distributions with 6, 7, 9 or 10 states are visually non-distinguishable. Looking at the relevant metrics reported in Table 4.20, we can see that the quality of the fitting substantially improves when we increase the number of phases from 2 to 3. With a 4-phase distribution, the second moment of the fitted distribution approaches greatly the empirical second moment. According to the quality aimed, one can approximate the data with a 3-state, a 4-state or a 6-state PH distribution.

Table 4.18: Number of tasks in the system

	Google Cluster data	Subcluster-5a	Subcluster-5b	
Tasks under service	average	126,408	37.4355	35.4433
	minimum	99,631	21	27
	maximum	171,232	70	66
Tasks waiting	average	6,004.22	0.861681	0.278734
	minimum	39	0	0
	maximum	90,548	17	49



(a) Subcluster-5a (b) Subcluster-5b

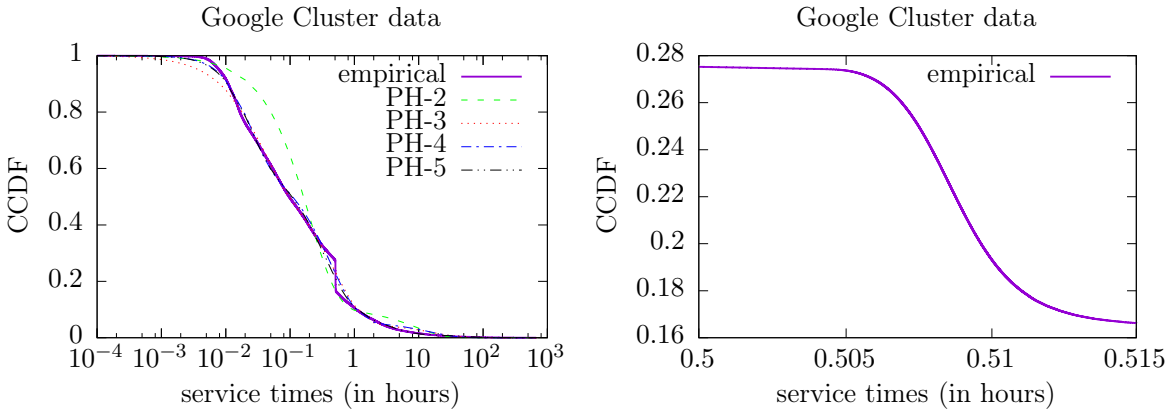
Figure 4.19: Delays observed before resubmissions during 29 days logging period for two subclusters.

Table 4.19: Google cluster: Relevant metrics of the service times

	1st moment	2nd moment	Log-likelihood
Trace	1.10 h	90.1 h^2	–
PH-2	1.10 h	16.7 h^2	– 0.033691
PH-3	1.10 h	28.0 h^2	0.158648
PH-4	1.10 h	25.6 h^2	0.186673
PH-5	1.10 h	60.6 h^2	0.214686
PH-6	1.10 h	60.6 h^2	0.214686
PH-7	1.10 h	63.1 h^2	0.219688
PH-8	1.10 h	63.3 h^2	0.232580
PH-9	1.10 h	63.4 h^2	0.238442
PH-10	1.10 h	86.5 h^2	0.243826

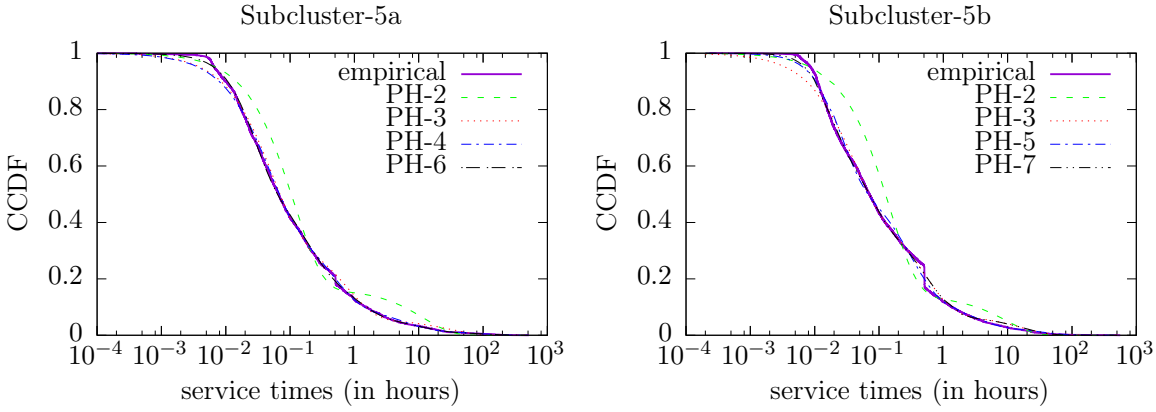
4.7.1.3 Subcluster-5b

We have 7,449 samples of service time that we use in the fitting. The CCDFs of empirical and some of the fitted distributions are depicted in Figure 4.21b. The best fit is obtained with 7 phases. Looking at the relevant metrics reported in Table 4.21, we see again that the



(a) Empirical and fitted distributions (b) Zoom around 0.5 (in hours)

Figure 4.20: Google cluster data: CCDF of empirical and fitted distributions of the service times.



(a) Subcluster-5a (b) Subcluster-5b

Figure 4.21: CCDF of empirical and fitted distributions of the service times.

Table 4.20: Subcluster-5a: Relevant metrics of the service times

	1st moment	2nd moment	Log-likelihood
Trace	2.06 h	240 h^2	–
PH-2	2.06 h	47 h^2	0.033000
PH-3	2.06 h	112 h^2	0.229758
PH-4	2.06 h	224 h^2	0.251971
PH-5	2.06 h	141 h^2	0.270744
PH-6	2.06 h	248 h^2	0.280319
PH-7	2.06 h	248 h^2	0.280312
PH-8	2.06 h	234 h^2	0.281664
PH-9	2.06 h	250 h^2	0.283754
PH-10	2.06 h	250 h^2	0.285382

Table 4.21: Subcluster-5b: Relevant metrics of the service times

	1st moment	2nd moment	Log-likelihood
Trace	1.47 h	133.0 h^2	–
PH-2	1.47 h	25.6 h^2	0.011947
PH-3	1.47 h	42.9 h^2	0.221906
PH-4	1.47 h	39.2 h^2	0.250760
PH-5	1.47 h	68.0 h^2	0.274209
PH-6	1.47 h	44.3 h^2	0.273130
PH-7	1.47 h	43.6 h^2	0.291051
PH-8	1.47 h	72.7 h^2	0.307154
PH-9	1.47 h	72.7 h^2	0.317256
PH-10	1.47 h	68.2 h^2	0.323152

quality of the fitting substantially improves when we increase the number of phases from 2 to 3. The second moment of the fitted distribution improves substantially as 5 phases are used instead of 4. According to the quality aimed, one can approximate the data with a 3-state, a 5-state or a 7-state PH distribution.

4.7.1.4 Fitting the arrival process of Google cluster data

Our processing of a task event table produces a set of records each contains an interarrival duration and the size of the arriving batch. We present next our analysis of the three traces corresponding to the three considered cases.

The trace of the arrival process contains 671,414 records featuring 751 distinct batch sizes. The analysis of the arrival process in the Google Cluster data presented in Section 4.5 shows that there are correlations between the interarrivals and 25% of the arrivals consist of more than 1 task and 5% consist of more than 100 tasks, the largest batch size observed being 90,050.

These first observations support our choice of modeling the arrival process as a BMAP. To perform the BMAP fitting, we used again the SPEM-FIT tool. This tool uses an Expectation-Maximization algorithm to estimate the parameters of the BMAP [Horváth 2013a]. We provide the tool with a trace containing the interarrivals with the corresponding batch sizes. Observe that due to the large sample size of the entire trace (671,414), the SPEM-Fit tool is able to return the optimal fitting only when the number of phases is less than 9.

Running the SPEM-Fit tool, we obtain an output file containing the matrices $\{Dk\}_k$ that characterize the fitted BMAP. Reading the output file in Python, we can use libraries of BUTOOLS [4] to derive the marginal distribution of the interarrivals. For the marginal distribution of the batch sizes, we do the following computation, for all possible values of batch sizes $k > 0$:

$$P(\text{batch size is } k) = \sum_{i=1}^{A} \pi_i \frac{\sum_{j=1}^{A} d_{ij}^{(k)}}{-d_{ii}^{(0)}}. \tag{4.1}$$

Here, π_i is the stationary probability of arrival phase i in the discrete-time Markov chain embedded at the jump times of the arrival phase process $J(t)$. Recall that for $k > 0$, $d_{ij}^{(k)}$ represents the rate of transitions from phase i to phase j accompanied by the arrival of a batch of size k, and $-d_{ii}^{(0)} = \sum_{j \neq i} dij^{(0)} + \sum_k \sum_j d_{ij}^{(k)}$, where $d_{ij}^{(0)} \leq 0$ with $j \neq i$ represents the transition rate from phase i to phase j without any arrival.

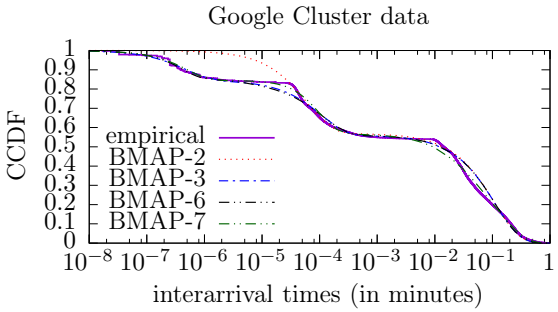
By noting $\tilde{\pi}_i$ the stationary distribution of the arrival phase at jump times and a diagonal matrix having the diagonal elements of matrix D_0, we can rewrite the batch size marginal distribution as follows:

$$P(\text{batch size is } k) = \tilde{\boldsymbol{\pi}}_{\boldsymbol{i}}(-\boldsymbol{B})^{(-\mathbf{1})}\boldsymbol{D_k} \tag{4.2}$$

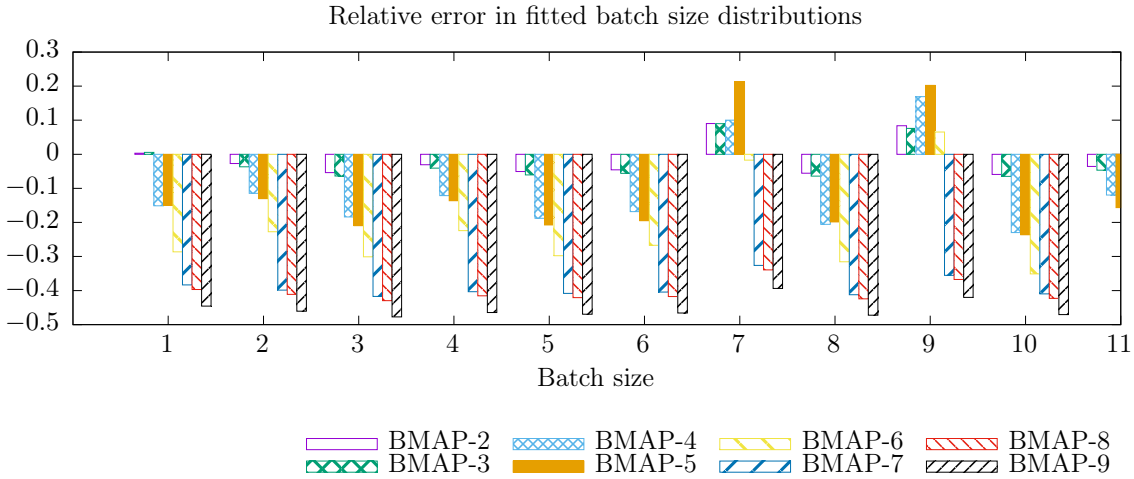
where $\mathbf{1}$ is a properly sized column vector having all elements equal to 1.

We report in Figure 4.26a the relative error between the probabilities that the two probability distributions, the empirical one of and the fitted one in Figure 4.22a, can assign to the same batch size. Only the results of the five first batch sizes are displayed, and these indicate that the fitted BMAP with two or three phases matches the empirical distribution better than those with a higher number of phases. To better assess which fitted distribution estimates best the empirical one we compute various statistical distances and provide the results in Table 4.23. The *total variation distance* is the maximum difference between probabilities assigned to any event by the two distributions. It ranges between 0 and 1 and a smaller value indicates that the fitted distribution estimates better the empirical one. The *Hellinger distance* is defined as the L2 distance between the square-roots of the two distributions. Again, the closer two distributions are, smaller is their Hellinger distance. The *Bhattacharyya distance* ranges between 0 and ∞ and is the logarithm of the inverse of the Bhattacharyya coefficient that ranges from 0 to 1. A coefficient of 1 indicates a perfect match between the two distributions, hence the smaller the distance the better. The minimum distances are reported in bold in Table 4.23 and confirm our earlier observation about BMAP-2 and BMAP-3. As far as the batch size distribution is concerned, the best is to fit the arrival process with a 3-phase BMAP. We now consider the marginal distribution of the interarrivals. We depict in Figure 4.22a the CCDFs of the empirical distribution and of some of the fitted distributions. The marginal distributions for 3 to 6 phases are very close. Similarly, the CCDF of the fitted distribution with 8 and 9 states are visually non-distinguishable from the one with 7 states. A similar observation can be made when comparing the relevant metrics reported in Table 4.25: the figures reported in the last three rows (states 7-9) are very close. We recommend using the 7-phase BMAP for the interarrivals if the quality of the fitting is the primary objective. But if one seeks to reduce the complexity, then a 3-phase BMAP is a

[4] Available at `https://github.com/ghorvath78/butools`.



(a) CCDF of empirical/fitted distributions of interarrivals



(b) Relative error between probabilities of empirical/fitted PMFs

Figure 4.22: Google cluster data: distributions of the batch size and of the interarrivals.

convenient choice (see Figure 4.22a).

4.7.1.5 Fitting the subclusters' interarrival times

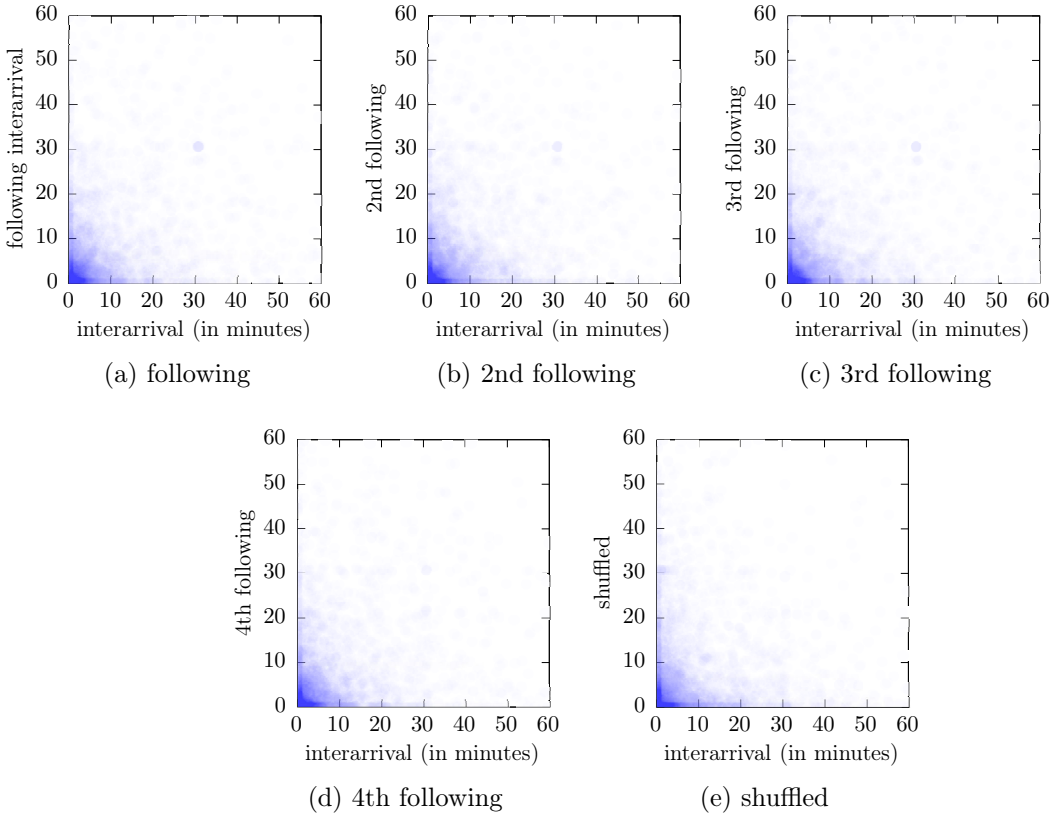
Before performing the BMAP fitting, we first investigate whether correlations are still present in the subtraces that we built. As each subtrace contains the informations only about tasks that were scheduled on five specific machines, we expect little correlations between interarrivals. The scatter plots shown in Figures 4.23-4.24 confirm our intuition. As for the case of the entire trace, each of these figures has five scatter plots. In Figures 4.23a and 4.24a, we depict the relation between any interarrival and its following one. In Figures 4.23e and 4.24e, we show the relation between shuffled interarrivals. By shuffling interarrivals, we remove any possible correlation between the considered interarrivals. In between, we show the relation between an interarrival and the 2nd/3rd/4th interarrival following it. All five scatter plots of Figure 4.23 are similar except for the bottom left corner: the density of points is higher in Figures 4.23a-4.23d than in Figure 4.24d indicating that interarrivals of few minutes are likely to be followed by intervals in the same range. A similar observation can be made on Figure 4.24 the scatter plot with shuffled interarrivals has a smaller density in the bottom

Table 4.22: Google cluster data: Relevant metrics of the interarrivals

	1st moment	2nd moment	Lag-1 joint moment	Log-likelihood
Trace	3.7318 s	61.5132 s^2	19.0058 s^2	–
BMAP-2	3.7318 s	48.8880 s^2	16.8858 s^2	2.48144
BMAP-3	3.7318 s	49.6908 s^2	16.8271 s^2	3.08328
BMAP-4	3.7318 s	49.6908 s^2	16.8275 s^2	3.11650
BMAP-5	3.7318 s	50.0940 s^2	16.7926 s^2	3.14762
BMAP-6	3.7318 s	49.6728 s^2	16.8188 s^2	3.19399
BMAP-7	3.7318 s	58.1040 s^2	18.4111 s^2	3.25296
BMAP-8	3.7318 s	58.1004 s^2	18.4122 s^2	3.26999
BMAP-9	3.7318 s	58.1472 s^2	18.4302 s^2	4.28787

Table 4.23: Google cluster data: Distances between empirical and fitted batch size distributions

Distance	BMAP-2	BMAP-3	BMAP-4	BMAP-5	BMAP-6	BMAP-7	BMAP-8	BMAP-9
Total variation	**0.007051**	0.008466	0.076099	0.078027	0.136973	0.192358	0.199016	0.223530
Hellinger	0.030453	**0.014738**	0.080431	0.082676	0.150191	0.215822	0.224352	0.256589
Bhattacharyya	0.000799	**0.000109**	0.073363	0.073706	0.154717	0.242897	0.253839	0.296318



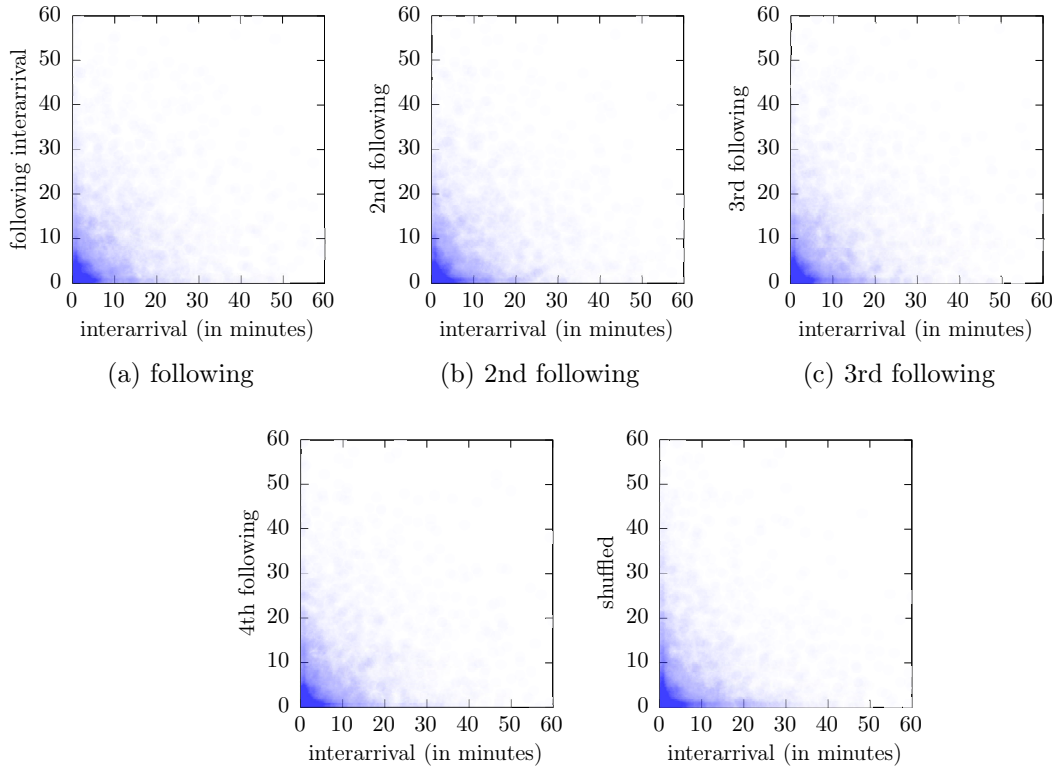
(a) following (b) 2nd following (c) 3rd following

(d) 4th following (e) shuffled

Figure 4.23: Subcluster-5a: Interarrivals are on the x-axis and on the y-axis we have (a) following interarrivals, (b) second following interarrivals, (c) third following interarrivals, (d) fourth following interarrivals, and (e) shuffled interarrivals.

left corner when compared to the other scatter plots.

Subcluster-5a The trace of the arrival process contains 3,724 records featuring batch sizes ranging from 1 to 5. The probability mass function of the batch size is listed in columns 1-2 of



(a) following (b) 2nd following (c) 3rd following

(d) 4th following (e) shuffled

Figure 4.24: Subcluster-5b: Interarrivals are on the x-axis and on the y-axis we have (a) following interarrivals, (b) second following interarrivals, (c) third following interarrivals, (d) fourth following interarrivals, and (e) shuffled interarrivals.

Table 4.24: Subcluster-5a: Distances between empirical and fitted batch size distributions

Distance	BMAP-2	BMAP-3	BMAP-4	BMAP-5	BMAP-6	BMAP-7	BMAP-8	BMAP-9	BMAP-10
Total variation	02415	0.02150	0.00434	**0.00912**	0.01418	0.01469	0.01203	0.05051	0.00990
Hellinger	0.05184	0.04559	**0.01459**	0.02310	0.03864	0.03871	0.03490	0.06520	0.03830
Bhattacharyya	0.00134	0.00104	**0.00011**	0.00027	0.00075	0.00075	0.00061	0.05268	0.00540

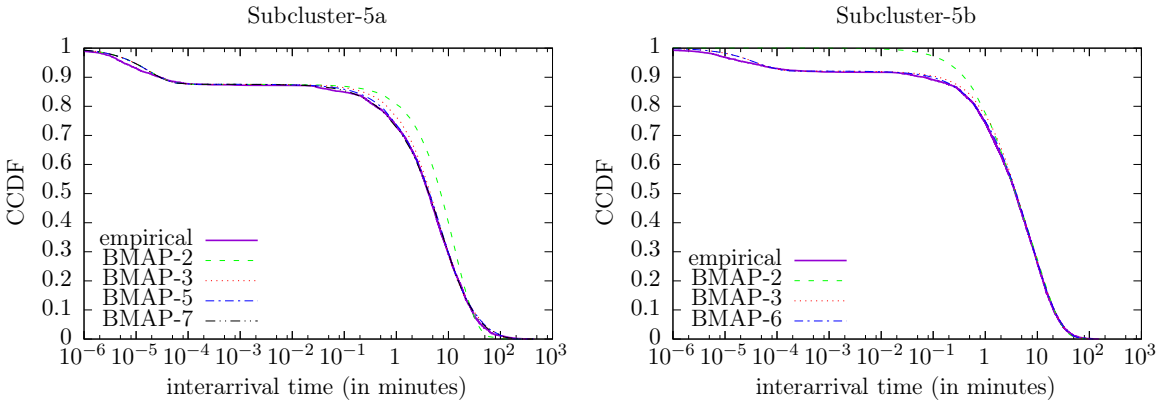
Table 4.26. We use again SPEM-Fit to fit a BMAP into the empirical arrival process. Recall that we provide the tool with a set of records reporting each an interarrival and a batch size, records being ordered chronologically. We considered up to 10 phases in the fitting procedures. Given the matrices $\{D_k\}_k$ returned by SPEM-Fit, we compute the marginal distribution of the batch size using (4.2). We also compute the marginal distribution of the interarrivals using libraries of BUTOOLS written in Python. We next evaluate the quality of the fitting with respect to the batch size. We report in Figure 4.26 the relative error between the probabilities that the two probability distributions, the empirical one of Table 4.26 and the fitted one in 4.2, can assign to the same batch size. The results hint that the fitted BMAP with four phases could match the empirical distribution better than those with a lower/higher number of phases. To confirm this observation we compute three statistical distances between each fitted distribution and the empirical one and report them in Table 4.28. The values in bold correspond to the minimal value for each distance and are obtained with BMAP-4.

As concerns the interarrivals, the quality of the fitting results can be assessed through

Table 4.25: Subcluster-5a: Relevant metrics of the interarrivals

	1st moment	2nd moment	Lag-1 joint moment	Log-likelihood
Trace	11.2050 min	664.524 min^2	251.053 min^2	-
BMAP-2	11.2026 min	286.700 min^2	128.938 min^2	1.55261
BMAP-3	11.2062 min	500.184 min^2	221.634 min^2	1.77016
BMAP-4	11.2050 min	509.292 min^2	226.332 min^2	1.80697
BMAP-5	11.2044 min	514.728 min^2	229.946 min^2	1.81996
BMAP-6	11.2038 min	534.492 min^2	240.343 min^2	1.83252
BMAP-7	11.2050 min	555.696 min^2	249.851 min^2	1.84070
BMAP-8	11.2056 min	560.880 min^2	251.705 min^2	1.84534
BMAP-9	11.2044 min	531.504 min^2	238.475 min^2	1.85549
BMAP-10	11.2056 min	558.036 min^2	251.269 min^2	1.85477

the figures reported in Table 4.27. A 2-state BMAP does not fit well the empirical arrival process as correlations are not well captured (the lag-1 joint moment of BMAP-2 is 128.938 min^2 while that from the trace is 251.053 min^2; see column 4 in Table 4.27). It requires to have at least a 3-state BMAP to approach the lag-1 joint moment. We plot the CCDFs of the empirical interarrivals and of some of the fitted distributions in Figure 4.25a. The marginal distributions for 4 to 7 phases are very close. Similarly, the CCDF of the fitted distribution with 9 or 10 states are visually non-distinguishable from the one with 8 states. Therefore, we recommend using the 4-phase BMAP for the interarrivals if the quality of the fitting is the primary objective. But if one seeks to reduce the complexity, then a 3-phase BMAP is a convenient choice.



(a) Subcluster-5a (b) Subcluster-5b

Figure 4.25: CCDF of empirical and fitted distributions of the arrival times.

Subcluster-5b The trace of the arrival process contains 5,223 records and batch sizes range from 1 to 6 (see distribution in Table 4.17). We repeat the same fitting procedure and will present the results through the Figures 4.25b and 4.26b and Tables 4.26 and 4.28.

Figure displays the relative error between the probabilities that the empirical distribution of Table 4.26 and the fitted distribution computed in 4.2 can assign to the same batch size. BMAP-4 appears to have the smallest relative error across the different batch sizes. Table 4.28 reports the total variation distance and the Hellinger and the Bhattacharyya distances

Table 4.26: Distribution of the batch size in two subclusters

Subcluster-5a		Subcluster-5b	
1	0.9242750	1	0.8680830
2	0.0585392	2	0.0882635
3	0.0120838	3	0.0319745
4	0.0045650	4	0.0097645
5	0.0005371	5	0.0015317
		6	0.0003829

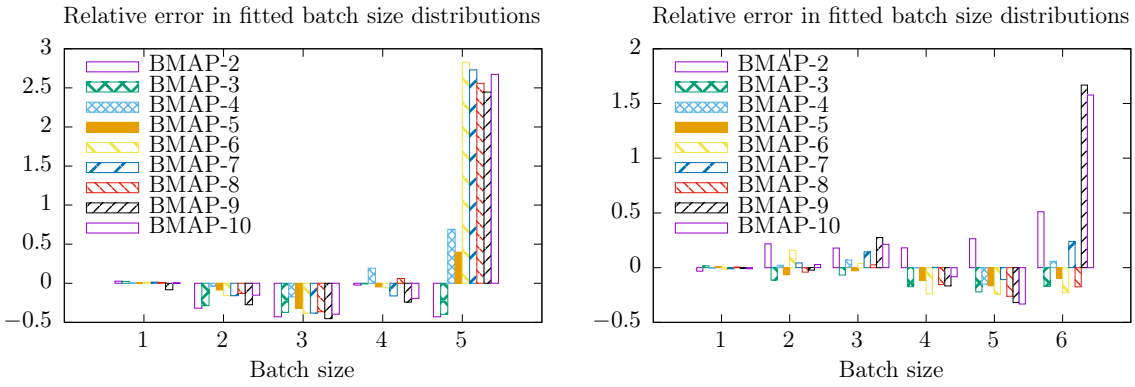
Table 4.27: Subcluster-5b: Relevant metrics of the interarrivals

	1st moment	2nd moment	Lag-1 joint moment	Log-likelihood
Trace	7.9956 min	200.448 min^2	91.0044 min^2	–
BMAP-2	7.9944 min	169.560 min^2	79.2180 min^2	0.68921
BMAP-3	7.9944 min	174.845 min^2	80.2980 min^2	1.27139
BMAP-4	7.9938 min	179.831 min^2	83.1204 min^2	1.30153
BMAP-5	7.9938 min	179.035 min^2	82.5084 min^2	1.32316
BMAP-6	7.9938 min	181.087 min^2	82.7280 min^2	1.32947
BMAP-7	7.9938 min	181.055 min^2	83.0700 min^2	1.33811
BMAP-8	7.9932 min	192.989 min^2	89.4384 min^2	1.35109
BMAP-9	7.9932 min	193.082 min^2	89.3376 min^2	1.35734
BMAP-10	7.9932 min	197.640 min^2	91.4472 min^2	1.35998

between the marginal batch size distributions of the fitted BMAP and empirical data. The results confirm that BMAP-4 brings the best estimate for the marginal distribution of batch sizes. Figure 4.25b depicts the marginal distribution of the interarrivals in the fitted BMAPs and of the empirical data. We observe that at least 3 states are needed to have a BMAP that fits well the empirical interarrivals.

Table 4.28 collects relevant fitting metrics. To better capture the correlations (more precisely, to match the lag-1 joint moment), an 8-state BMAP can be used at the cost of higher complexity.

For our future numerical simulations, we recommend to use 3 states for the BMAP fitting and 3 states for the PH fitting of the service time. The system is then modeled by a 5-



(a) Subcluster-5a (b) Subcluster-5b

Figure 4.26: Relative error between probabilities of empirical/fitted marginal batch size distributions.

Table 4.28: Subcluster-5b: Distances between empirical and fitted batch size distributions

Distance	BMAP-2	BMAP-3	BMAP-4	BMAP-5	BMAP-6	BMAP-7	BMAP-8	BMAP-9	BMAP-10
Total variation	0.02730	0.01437	**0.00415**	0.00800	0.01517	0.00857	0.00557	0.00943	0.00995
Hellinger	0.03894	0.02258	**0.00784**	0.01290	0.02754	0.01507	0.01207	0.02879	0.02406
Bhattacharyya	0.00076	0.00026	**0.00003**	0.00008	0.00038	0.00011	0.00007	0.00041	0.00029

Table 4.29: Nef cluster data: Parameters extracted from the traces after processing

Task event table	Nef Cluster data
Batch sizes	533
Tasks arrival rate	23.7015 s^{-1}
Average service time	45.0252 min
Resubmission probability	0.0032
Load	31.68
Average impatience time	0.0305 s
Abandonment rate	0.000074

dimensional Markov chain. As for the value of c, numerical results for multiple values should be derived and the value returning the best match between analytical and empirical results should be adopted. A starting point could be to take c between the average count of tasks under service and the maximum count. In other words, $c = 54$ for Subcluster-5a and c = 50 for Subcluster-5b.

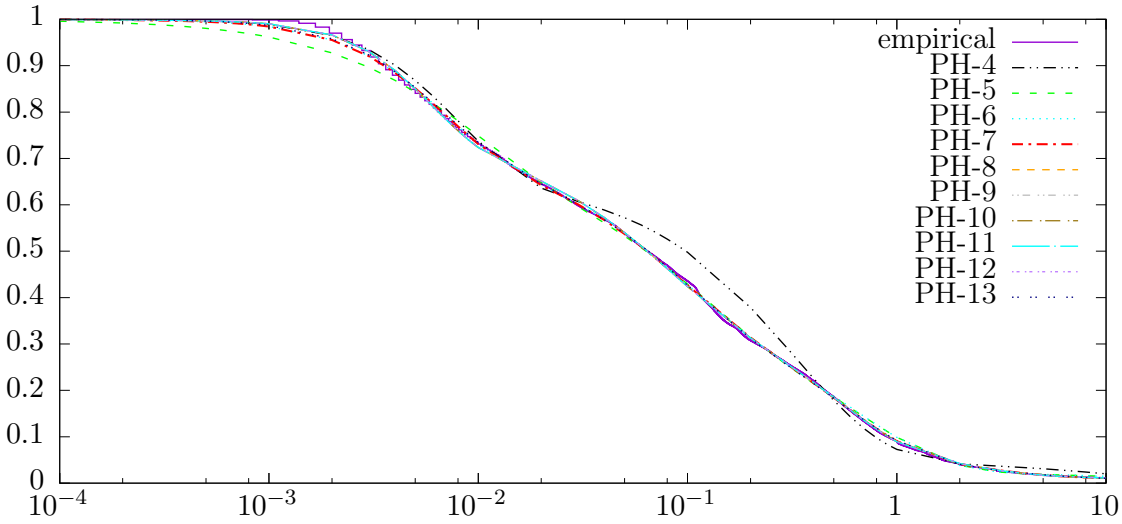
4.7.2 Nef traces

After processing the Nef traces, we present the extracted parameters in Table 4.29. Based on the load, the system is stable as long as there are at least 32 servers, thus $c = 32$ The fitting of the arrival and service process are discussed in the upcoming sections.

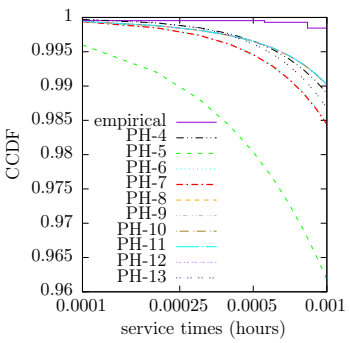
4.7.2.1 PH Fitting of the service times

We have 5,000,364 samples of service time that we use in the fitting. The complementary cumulative distribution functions (CCDF) of empirical and fitted distributions are depicted in Figure 4.27. The CCDF of the fitted distribution with more than 17 states are visually non-distinguishable from the one with 17 states. The CCDF of the fitted distribution with 7, 8 and 9 states are not visually distinguishable among them. Hence, only the 7-state distribution is presented in Figure 4.27. To distinguish visually the CCDF fitted distributions, we depict in Figures 4.27c, 4.27b, 4.27d the zoom in on some specific regions from Figure 4.27.

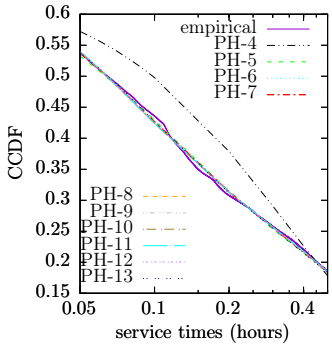
Looking at the relevant metrics reported in Table 4.30, we can see that the quality of the fitting substantially improves when we increase the number of phases from 4 to 5 and then from 6 to 7. Next, the quality of fitting decreases for phase 8 and then remains stable to phase 10. Then, it decreases until the phase 12 and then for phase 13 increases again. From phase 17 to 20, it remains stable. For phase 13, we have the second moment of the fitted distribution which approaches greatly the empirical second moment. According to the quality aimed, one can approximate the data with a 13-state and a 17-state PH distribution. Aiming to minimize the complexity, we do not propose a 18-state, a 19-state and a 20-state PH distributions, since the second time moment is exactly the same. If one seeks to



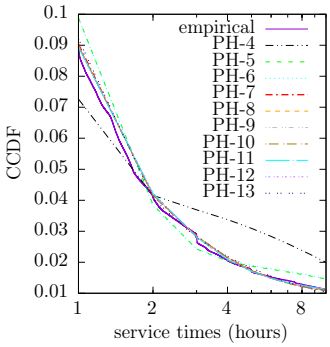
(a) Empirical and fitted CCDF of the service times



(b) Zoom around 1 second



(c) Zoom around 0.15 hours



(d) Zoom from 1 until 10 hours

Figure 4.27: Nef data cluster: (a) CCDF of the empirical and fitted distributions of service times, (b) Zoom around 1 second (c) Zoom around 0.15 hours and (d) Zoom from 1 until 10 hours.

minimize essentially the complexity, then a 7-state PH distribution is a good tradeoff between complexity and fitting quality.

4.7.2.2 BMAP Fitting of the interarrival times

We fit the arrival process of Nef cluster as BMAP. The relevant metrics of the interarrivals are presented in Table 4.31. Observe that the best fitting is for the number of states 12, 8 or 3. Aiming at reducing the complexity, then a 3-phase BMAP is the best choice. Even though, none of the second moments of fitting distributions are close to the empirical ones. An explanation could be that the interrarrivals have correlations as discussed in Section 4.5.2.

4.8 Discussion

In this chapter, we presented a global analysis for two different types of data center cluster workload. For these workload datasets, we characterized the abandonment, the arrival, the

Table 4.30: Nef cluster data: Relevant metrics of the service times

	1st moment	2nd moment	Log-likelihood
Trace	0.75 h	30.8 h^2	–
PH-2	0.75 h	1.3 h^2	$-\infty$
PH-3	0.75 h	14.3 h^2	0.613688
PH-4	0.75 h	12.7 h^2	0.627025
PH-5	0.75 h	20.6 h^2	0.65147
PH-6	0.75 h	20.3 h^2	0.685143
PH-7	0.75 h	**27.4 h^2**	0.691204
PH-8	0.75 h	27.1 h^2	0.69628
PH-9	0.75 h	27.1 h^2	0.69628
PH-10	0.75 h	27.1 h^2	0.696279
PH-11	0.75 h	26.5 h^2	0.696383
PH-12	0.75 h	26.3 h^2	0.696398
PH-13	0.75 h	**29.7 h^2**	0.69426
PH-14	0.75 h	29.5 h^2	0.696189
PH-15	0.75 h	22 h^2	0.696706
PH-16	0.75 h	26.9 h^2	0.703459
PH-17	0.75 h	32 h^2	0.704377
PH-18	0.75 h	32 h^2	0.704381
PH-19	0.75 h	32 h^2	0.706963
PH-20	0.75 h	32 h^2	0.704471

service and the waiting times. As far as the abandonment distributions are concerned, we showed that the best fitting corresponds to a Burr distribution, even though Weibull and exponential can be an initial choice. For each case, we validated that the arrival process can be modeled as a Batch Markovian Arrival Process and proposed one and then proposed a phase-type distribution for the service process. In order to decrease the complexity of our model, we chose the smallest value of the number of phases among the ones that produced valid results. We then validated and quantified our model features which were introduced in Chapter 3; the abandonment ratio and the resubmission probability. As a preliminary assessment of the results, there does not seem to be any other feature that should be taken into account. The fitting of the Nef arrival process does not perform as well as in the Google traces case. This can be justified by the fact that there are many interarrival times which are the same, and there are too few "training" samples for many arrival types. To provide some intuition for why this happens, it should be noted that the timestamps regarding the interarrival times in the Nef traces cluster case are given in seconds, whereas in the Google traces case they were given in microseconds, increasing the dataset variability in the second case.

During the above analysis, we attacked the following difficulties. First, using Hadoop, we managed to overcome the technical challenge related to parsing the 500 files each time and gathering all the information for one job in one file. Secondly, to overcome the technical difficulty related to the number of servers, c, that can be used in our numerical simulations, we extracted two different subclusters for the Google case and showed that the second one seems capable of representing the whole trace. Due to time restrictions, the subcluster extraction process was not performed for the Nef cluster, but it would be interesting to explore in future

Table 4.31: Nef Cluster data: Relevant metrics of the interarrivals

	1st moment	2nd moment	Log-likelihood
Trace	1.5052 s	3.1611 s^2	–
BMAP-2	1.5052 s	0.4890 s^2	9.66388
BMAP-3	1.5052 s	0.6696 s^2	9.59515
BMAP-4	1.5052 s	0.3629 s^2	9.20879
BMAP-5	1.5052 s	0.3585 s^2	9.32180
BMAP-6	1.5052 s	0.3585 s^2	9.39845
BMAP-7	1.5052 s	0.6808 s^2	9.45206
BMAP-8	1.5052 s	0.6687 s^2	9.51056
BMAP-9	1.5052 s	0.6644 s^2	9.55692
BMAP-10	1.5052 s	0.6696 s^2	9.59515
BMAP-11	1.5052 s	0.6618 s^2	9.62777
BMAP-12	1.5052 s	6.6874 s^2	9.65552
BMAP-13	1.5052 s	0.5391 s^2	9.68303
BMAP-14	1.5052 s	0.5365 s^2	9.70819
BMAP-15	1.5052 s	0.5331 s^2	9.73082

work.

Chapter 5

marmoteCore-Q: A tool for Simulating Queueing Models

Contents

In this chapter, we describe the marmoteCore-Q tool that can be used to simulate numerically complex queues but also simpler queueing systems with known explicit closed form or specific numerical algorithms. This tool can be used to compute the solution to the complex model BMAP/PH/c/N+c queue model with impatience and reentry as presented in Chapter 3, overcoming the implementation challenges. To validate the BMAP/PH/c/N+c solution, we provide the validation methodology and some numerical test-cases.

Keywords: Queueing models, Simulations, BMAP/PH/c/N+c

5.1 Implementation details

In this section, we describe the implementation details, which include the data structures, the main programming steps, the input parameters and the program output.

For our development of marmoteCore-Q tool, the marmoteCore platform is necessary. For this reason, we first need to download and install marmoteCore, which is implemented in C/C++, from [b]. More installation details can be found in Appendix E.

In the following subsections, we first present our data structure in Section 5.1.1. We then discuss, in Section 5.1.2 the main programming steps and summarize the main and auxiliary functions. In Section 5.1.3, we explain how we compute the infinitesimal generator Q. Finally, we discuss, in Section 5.1.4, the marmoteCore-Q input parameters and the output files in Section 5.1.5.

5.1.1 Data Structure: problemData

Before we present the tasks performed, we explain below our data structure. For efficient access and modification, we use this data structure to organize, manage and store the data of our model. Our data structure is presented below:

```
struct problemData {
       int servers;
       int N;
       double a;
       double ps;
       double lam;
       double miu;
       int Q_car;

       double ***arrival;
       double ***service;
       int phases;
       int vt;
       int K;
       Simplex* f;
       Simplex* f1;
       int cardinals;
       int cardinals0;
       double *t0;
}pbm;
```

and each structure member corresponds respectively to:

- number of servers;
- queue capacity;
- abandonment rate α;
- resubmission probability p_s;
- arrival rate λ;
- service rate μ;
- Q matrix dimensions;
- matrix of the representation of service process: T, β, the parameters of the service time distribution. T a transition structure and β is a vector of a probability distribution;
- matrices of the representation of BMAP: $D_0, ..., D_K$, where K is the maximum batch size;
- number of arrival phases;

- number of service phases;
- maximum batch size, K;
- Simplex object f, which represents the sequence of integer numbers when the total sum is c;
- cardinals corresponds to $\binom{server+phases-1}{phases-1}$;
- cardinals0 corresponds to $\binom{server+phases}{phases}$;
- $T_0 = -T\mathbf{1}$;

5.1.2 Programming Steps

We present below the tasks performed:

- create the infinitesimal generator Q of the queue model from the basic parameters. To create the infinitesimal generator Q, we create a SparseMatrix object to hold the transition matrix of the chain, entry by entry with the addToEntry() function and we give as input parameter the structure which is described above (see Section 5.1.1);

  ```
  SparseMatrix* Q_infinit_gen(struct problemData *pbm);
  ```

- create a DiscreteDistribution object to hold the initial distribution of the process;
- set the type of the infinitesimal generator Q to continuous

  ```
  Q->set_type( CONTINUOUS );
  ```

- create a MarkovChain object and link the infinitesimal generator Q to it;

  ```
  MarkovChain* myMC = new MarkovChain( Q );
  ```

- uniformize the continuous Markov Chain that we created in the previous step;

  ```
  MarkovChain* uMC = myMC->Uniformize();
  ```

- convert type (casting) from TransitionStructure* to SparseMatrix* and get the generator of uMC, with the aid of function

  ```
  uMC->generator();

  SparseMatrix *spQ = dynamic_cast<SparseMatrix*>(uMC->generator());
  ```

- produce diagnostics for the generator such that

  ```
  spQ->Diagnose( stdout );
  ```

  The SparseMatrix structure corresponds to the infinitesimal generator Q of our model. The diagnostic for a SparseMatrix structure are:

- generator type
  The generator types are either continuous or discrete.
- number of origin states of infinitesimal generator Q
- number of destination states of infinitesimal generator Q
- number of transitions of infinitesimal generator Q
- number of empty rows: Let us define $max\{v_i\}$ and $min\{v_i\}$ as the maximum value and the minimum value of the row i of infinitesimal generator respectively. If the $max\{v_i\} < min\{v_i\}$, then the row is considered as empty.
- maximum outdegree: Let out_i be the number of outgoing links of node i, then $outdegree_{max} = max\{out_i\}_{i\in[1,...,C]}$, where C is the number of states of a Markov Chain.
- minimum outdegree: Let out_i be the number of outgoing links of node i, then $outdegree_{min} = min\{out_i\}_{i\in[1,...,C]}$, where C is the number of states of a Markov Chain.
- maximum indegree: Let in_i be the number of ingoing links of node i, then $indegree_{min} = min\{in_i\}_{i\in[1,...,C]}$, where C is the number of states of a Markov Chain.
- minimum indegree: Let in_i be the number of ingoing links of node i, then $indegree_{max} = max\{in_i\}_{i\in[1,...,C]}$, where C is the number of states of a Markov Chain.
- maximum value of the infinitesimal generator Q: Let us define the maximum value as v_{max} which is at most 1 since the infinitesimal generator represents transition probabilities.
- minimum value of the infinitesimal generator Q, v_{min}: Let us define the maximum value as v_{min}.
- maximum row sum: Let us define $S_i = \sum_j Q_{ij}$ and compute $\min_{i=\{1,...,C\}} S_i$.
- minimum row sum: Let us define $S_i = \sum_j Q_{ij}$ and compute $\min_{i=\{1,...,C\}} S_i$.
- row sum mismatch: Let us define x as a flag of row mismatch:

$$x = \begin{cases} 0, \text{ if } \max\limits_{i=\{1,...,C\}} S_i = \min\limits_{i=\{1,...,C\}} S_i \\ \\ 1, \text{ if } \max\limits_{i=\{1,...,C\}} S_i \neq \min\limits_{i=\{1,...,C\}} S_i \end{cases} \tag{5.1}$$

- apply the iterative standard Power method on probability transition matrix

```
Distribution *stDis0 = uMC->StationaryDistributionIterative( "Power"
    ,10000, prec, "Zero",\\ NULL, /* progress? */ true);
```

`prec`: A precision parameter is used as the stopping criterion for the iterative computation. This means that the computation stops as soon as two consecutive iterations are 'closer' to each other than this preset parameter.

- `DiscreteDistribution *stDis = dynamic_cast<DiscreteDistribution*>(stDis0);`

- open the file and write to it the stationary distribution function in default format

```
FILE* out = fopen( fileName.c_str(), "w" );
stDis->Write(out, DEFAULT_PRINT_MODE);
```

- open the file and write to it the stationary distribution function in maple format. Details on why maple format is useful are presented in Section 5.2.1.3.

```
FILE* out1 = fopen( fileName1.c_str(), "w" );
stDis->Write(out1, MAPLE_PRINT_MODE);
```

- open output file with the name of taking its name from the input parameter

```
string fileName2;
fileName2 = title + ".txt";
ofstream outputFile;
outputFile.open(fileName2.c_str());
```

- compute agregate load statistics such that customer arrival and service rate, and, offered load

- compute and save the marginal of the queue size (a distribution over $\{0, 1, ..., N - c\}$) to output file

```
double marginal_queue_size(ofstream &outputFile, DiscreteDistribution*
    dis, struct problemData *pbm)
```

- compute and save the marginal of the number of busy servers (a distribution over $\{0, 1, ..., c\}$) to output file

```
void number_busy_server_distrib(ofstream &outputFile,
    DiscreteDistribution* dis, struct problemData *pbm)
```

- for each phase, compute and save the distribution of the number of servers in that phase (S distributions over {0,1, ...,c}) to output file

```
void service_phase(ofstream &outputFile, DiscreteDistribution* dis,
    struct problemData *pbm)
```

- compute and save the marginal of the arrival phase (a distribution over $\mathcal{E}_A = \{1, ..., A\}$) to output file

```
void arrival_phase(ofstream &outputFile, DiscreteDistribution* dis,
    struct problemData *pbm)
```

- compute queue size statistics (e.g average queue size, moments and variance queue size), performance metrics like expected number of servers, expected waiting time and computations of flows like accepted, lost, served and reneged throughput.

Table 5.1: Implemented main functions.

Type	method	description
`long int`	`index0(int sum, struct problemData *pbm, int v, int* foo)`	return the position (line or column) of the infinitesimal generator Q
`int`	`index_fun(Simplex* f, int n, int* foo, struct problemData *pbm, int cardinals, int v)`	index for $n \leq 1$
`SparseMatrix*`	`Q_infinit_gen(struct problemData *pbm)`	compute infinitesimal generator
`double`	`marginal_queue_size(ofstream &outputFile, DiscreteDistribution* dis, struct problemData *pbm)`	see Section 3.4.6
`double`	`arrival_phase(ofstream &outputFile, DiscreteDistribution* dis, struct problemData *pbm)`	see Section 3.4.5
`void`	`number_busy_server(ofstream &outputFile, DiscreteDistribution* dis, struct problemData *pbm)`	see Section 3.4.7
`void`	`service_phase(ofstream &outputFile, DiscreteDistribution* dis, struct problemData *pbm)`	see Section 3.4.8
`void`	`compute_flows(ofstream &outputFile, DiscreteDistribution* dis, struct problemData *pbm)`	see Section 3.4.3

```
void compute_flows_stat_metrics(ofstream &outputFile,
   DiscreteDistribution* dis, struct problemData *pbm, int moments )
```

We present the main and auxiliary implemented functions and their description in Table 5.1 and Table 5.2 respectively.

5.1.3 Infinitesimal Generator (Q)

The infinitesimal generator Q is a sparse matrix [1]. Hence, we take advantage of the matrix sparsity and we compute the values, based on the formulas from Chapter 3, finding only the corresponding Q indices for non-zeros elements. Each line and each column corresponds to one Markov chain state which is represented by a vector. We first find the possible transition states and then the corresponding Q indices. Therefore, we have to overcome two implementation challenges:

1. to find a way to compute the index of some vector $(n_1, ..., n_k)$
2. the vector that corresponds to some index i of the infinitesimal generator Q.

The marmoteCore platform permits us to overcome the above challenges, using the Simplex class which represents sequences of non-negative integer numbers with a given total sum. We create an object f calling the constructor of the Simplex member class;

[1] A sparse matrix or sparse array is a matrix in which most of the elements are zero.

Table 5.2: Implemented auxiliary functions.

Type	method	description
`int`	`file_lines(char* filename)`	return number of file lines , excluding the 1st line
`double*`	`parseLine(std::string str, int &number_of_cols)`	return pointer to the array of line values
`double**`	`parseMatrix(std::string str, int &number_of_cols)`	return pointer to the infinitesimal generator Q values
`double***`	`read_file_and_store_matrices(char* filename, int K, int &number_of_cols, double *lambda)`	read file and store matrices
`long int`	`binomial (int n, int p)`	compute $\binom{n}{p}$
`long`	`double factorial(int q)`	compute factorial
`long double`	`compute_rateQ0(int* foo, int q, int phases, double **p)`	compute rate $\binom{n}{r1,\, r2,\, ..,\, rl} * \Pi_{l=1}^{S} p^{rl}$
`int`	`sum(int* foo, int phases)`	return $\sum_{i=0}^{ph} s_i$
`double*`	`compute_t0(double **t, int phases)`	compute $T_0 = -T\mathbf{1}$
`void`	`printfn(SparseMatrix* ptr, int num_of_rows, int num_of_cols)`	print infinitesimal generator (Q) to the screen

`Simplex(int k, int c);`

The parameter k represents the number of service phases and specifies the length of the sequence, the c is the number of servers and specifies the total sum. The index of state $(s_1, s_2, ..., s_k)$ is given by the Simplex member function

`int Index(int* s);`

and the vector of index i is given by the Simplex member function

`int DecodeState(i, int* s);`

The state space, which corresponds to the case that customers waiting in the queue ($n > 0$), is $S_{k,c} = \{(s_1, s_2, ..., s_k) \in \mathbb{N}^k \text{ such that } \sum_{i=1}^{k} s_i = c\}$. Its cardinal, M is:

$$M = |S_{k,c}| = \binom{c+k-1}{k-1} = \binom{c+k-1}{c}. \tag{5.2}$$

The state space, which corresponds to the case that no customers waiting in the queue ($n = 0$), is $S^0_{k,c} = \{(s_1, s_2, ..., s_k) \in \mathbb{N}^k \text{ such that } \sum_{i=1}^{k} s_i \leq c\}$. Its cardinal, M_0 is:

$$M_0 = |S^0_{k,c}| = \binom{c+k}{k}. \tag{5.3}$$

To have all the possible $\vec{s}$ and the corresponding indices for sum $\leq c$, we create a second object f1, calling again the constructor of the Simplex class and increasing the length of the sequence k by one;

`Simplex(int k+1, int c);`

Recall from Section 3.3 that the vectors $\vec{s}$ are ordered first according to $|\vec{s}|$, then for the same $|\vec{s}| = m$, they are ordered lexicographically, where $|\vec{s}| = \sum_{l=1}^{k} s_l$. Hence, we first define the function which compute the index of state $\{n, (s_1, ..., s_k)\}$ as $index(c, k; s_1, ..., s_k)$. We then compute the index of state $\{n, (s_1, ..., s_k), v\}$ of the infinitesimal generator Q as:

$indexQ(c, k; s_1, ..., s_k, n, v) =$

$$\begin{cases} \sum_{i=0}^{|\sum_{j=0}^{k} s_j|} \binom{i+k-1}{k-1} + A * index(c, k; s_1, ..., s_k) + (v-1), \text{ if } n = 0 & (5.4) \\ A * (M_0 + index(c, k; s_1, ..., s_k) + (n-1) * M) + v, \text{ if } n \geq 2 & (5.5) \end{cases}$$

where n is the number of customer waiting in the queue and v is the the arrival phase.

Algorithm 1 demonstrates the steps to compute the infinitesimal generator Q.

Algorithm 1 Compute infinitesimal generator Q

```
Input: problemData structure
Output: Q matrix
create the sparse matrix Q object
for each server do
    compute partial cardinals ▷ partial cardinals are defined as the cardinals with |s⃗| = c
    create the corresponding simplex object
    for each partial cardinals do
        find the vector & add its element
        if the summation of the element of the vector is equal to 0 then
            for each server do
                create a simplex object
                compute cardinals
                for each cardinal do
                    find the vector
                    add the elements of the vector, sum
                    for each arrival phase do
                        if sum! = 0 AND sum < B then
                            compute value d_i j^(k) (k choose r_1...r_s) Π_{l=1} Sp_l^r   ▷ (see Section 3.3)
                            add value to the specific position of the matrix
                            if sum == c then
                                for each batch size b do
                                    if b ≤ N AND b + c < B then
                                        compute value
                                        add value to the specific position of the matrix
                                    else if sum == 0 then
                                        compute value v = d_ij^(0)
                                        add value v to the specific position of the matrix
                                    else
                                        for each arrival phase from 1 to A do
                                            if sum == c then
```

for each batch size b **do**
  if $b \leq N$ **then** ▷ N buffer size
    compute $val = d_{ij}^{(b+c-m)} \binom{b-m}{r_1 \ldots r_s} \Pi_{l=1}^{S} p^{r_l}$ ▷ (see Section 3.3)
    add val to Q matrix
    for each phase ph_1 **do** possible transitions change service phase
      for each phase ph_2 **do**
        if $s_{ph_2}! = 0$ AND $s_{ph_1} < c$ **then**
          $s_{ph_2} \leftarrow s_{ph_2} - 1$ & $s_{ph_1} \leftarrow s_{ph_1} + 1$
          compute $val \leftarrow s_{ph_2} * (t_{ph_1 ph_2} + t_{ph_1} * p * p_{ph_2})$
          for each arrival phase from 1 until A **do**
            add val to Q matrix position
            $s_{ph_2} \leftarrow s_{ph_2} + 1$ & $s_{ph_1} \leftarrow s_{ph_1} - 1$ increase each position by $x = 1, ..., c - m$
  if $s_{ph_1} \leq c$ AND $sum \leq c$ **then**
    for each batch size b **do**
      $s_{ph_1}^{(new)} \leftarrow s_{ph_1} + b$
      compute difference r one by one element between initial service vector and new one
      if $sum_{\vec{s}^{(new)}} \leq c$ AND $sum_r < B$ **then**
        for each arrival phase i from 1 to A **do**
          for each arrival phase j from 1 to A **do**
            compute $val = d_{ij}^{(b+c-m)} \binom{c-m}{r_1 \ldots r_s} \Pi_{l=1}^{S} P^{r_l}$
            add val to matrix Q decrease by one each phase
            if $s_{ph_1}^{(new)}! = 0$ **then**
              compute $val = s_{ph_1} * t_{ph_1 0} * (1 - p)$ $n \leq 1$ find possibles states and positions
  for each cardinal **do**
    find the service vector
    for each service phase ph_1 **do**
      compute $val1 = s_{ph_1} * t_{0 ph_1} * (1 - p) * p_{ph_1}$
  for each arrival phase **do**
    add value to Q matrix

60: **for** each customer n waiting in the queue from 2 to N **do** $Q_{n,n+1}$
61: **for** each arrival phase from 1 to A **do**
62: **if** $n == 2$ AND $n - 1 < B$ **then**
63: **for** each batch size **do**
64: **if** $b < N$ **then**
65: add value to Q matrix
66: **for** each batch size **do**
67: **if** $n + b >= N$ **then**
68: add value to Q matrix
69: **for** each arrival phase i **do**
70: add $val = n * \alpha + val1$
71: **for** each arrival phase j **do**
72: **if** $n == 2$ AND $i! = j$ **then**
73: add value $d_{ij}^{(0)}$ to Q ▷ for $n = 1$
74: **if** $i! = j$ **then**
75: add value $d_{ij}^{(0)}$ to Q ▷ for $n \geq 2$
76: **for** each service phase ph_1 **do**
77: **if** $s_{ph_1}! = 0$ **then**
78: **for** each service phase ph_2 **do**
79: **if** $ph_1! = ph_2$ AND $s_{ph_2} < c$ **then**
80: $s_{ph_1} = s_{ph_1} - 1$
81: $s_{ph_2} = s_{ph_2} + 1$
82: compute value $vQ_{nn1} = s_{ph_1} * t_{0ph_1} * (1 - p) * p_{ph_2}$
83: compute value $vQ_{nn} = s_{ph_1} * (t_{ph_1ph_2} + t_{ph_10} * p * p_{ph_2})$
84: add value vQ_{nn1} to Q matrix
85: add value vQ_{nn} to Q matrix
86: **if** $n == 2$) **then**
87: add value $v * Q_{nn1}$ to Q matrix ▷ for $n = 1$
88: add value $v * Q_{nn}$ to Q matrix ▷ for $n = 1$
89: $s_{ph_1} = s_{ph_1} + 1$
90: $s_{ph_2} = s_{ph_2} - 1$
return Q

We finally implement the queue size, arrival and busy server marginals, and for each phase, the the distribution of the number of servers at that phase as defined in Sections 3.4.6, 3.4.5,

3.4.7 and 3.4.8 respectively.

5.1.4 Input parameters

In this section, we discuss the input parameters that the user should pass from the command line after the executable file `./marmoteCore-Q`.

In case that we are not sure how to use the marmoteCore-Q, we run the command `./marmotecoreQ --help` or `./marmotecoreQ -h`. We will see usage information and the following list of options we can use with the command:

```
./marmoteCoreQ --help

  -h, --help display this help and exit
                          -u, --usage display a short usage message and exit
                          <number_of_servers> int
                          <buffer_size> int
                          <abandonment_rate> double
                          <resubmission_probability> double
                          <arrival_fitting_file> python file matrices D0..Dk
                              and last line arrival rate lambda
                          <service_fitting_file> python file matrices {Teta}
                              and last line service rate mu
                          <experiment_title> text name of the experiment e.g
                              MM1K_medium_load
```

Listing 5.1: Command how to use the marmotecore-Q

If we miss one input or change the order of the inputs, the following message appears in the screen:

```
./marmoteCoreQ [--help] [--usage] <number_of_servers> <buffer_size> <
    abandonment_rate> <resubmission_probability> <arrival_fitting_file> <
    service_fitting_file> <experiment_title>
```

Listing 5.2: Command how to use the marmotecore-Q

The necessary parameters are presented in Listing 5.1. Most of them are self-explanatory; with regard to arrival and service fitting files, and experiment title, more information can be found below:

arrival_fitting_file in python: In this file, the matrices of the representation of Batch Markovian Arrival Process (BMAP): $D_0, ..., D_K$, where K is the maximum batch size are saved. The arrival rate λ is computed by modifying the equation (3.1) to:

$$\lambda = \pi(\sum_{k=1}^{K} kD_k)\mathbf{1} \tag{5.6}$$

where, π is the stationary distribution and $\mathbf{1}$ represents an $m \times 1$ vector with every element being 1. marmoteCore provides the Power Method to compute the stationary distribution π as already mentioned in Section 5.1.2. The arrival rate is saved to the last line of the file.

To compute the arrival rate λ, for user convenience, we propose below 3 ways: the first one launching ipython3, the second one in python and the last one in Matlab. In the example Listing 5.3, the maximum batch size is $K = 6$ and we generate the BMAP matrices using SpemFit [g]. BMAP matrices can be generated in both python and matlab format. The user can choose one of the following ways to compute the arrival rate, according to their language preference.

1. We first download BuTools from [a] and then launch ipython3. We finally run the following commands:

```
%run "~/Fitting/butools2/Python/BuToolsInit.py"
butools.verbose = True
butools.checkPrecision = 1e-9
%run "result-3-m"
Dm = [D0, D1, D2, D3, D4, D5, D6]
Q = SumMatrixList(Dm[0:])
l = len(Q)
pi = CTMCSolve(Q)
DDm = [D1, 2*D2, 3*D3, 4*D4, 5*D5, 6*D6]
QQ = SumMatrixList(DDm[0:])
customers_per_time = pi*QQ*np.matlib.ones((l,1))
batch_per_time = - pi*D0*np.matlib.ones((l,1))
```

Listing 5.3: ipython3 comments for computing arrival rate λ

2. We use the python file which is generated by Spemfit [g] and we add the following code:

```
import numpy.matlib as ml
import numpy as np
from scipy.linalg import eig

# matrix D0 ..Dk in python format as they are generated by BuTools
.....

# \sum_{0}^{k}D_{k}
# Q = D0 + D1 + ..+ Dk;
# example:
Q = D0 + D1 + D2 + D3 + D4 + D5 + D6; # sum_k D_k
# compute stationary distribution
S, U = eig(Q.T)
stationary = np.array(U[:, np.where(np.abs(S) < 1e-8)[0][0]].flat)
stationary = stationary / np.sum(stationary)

print (stationary)
# QQ = 1*D1 + ... + k*Dk;
#example:
QQ = 1*D1 + 2*D2 + 3*D3 + 4*D4 + 5*D5 + 6*D6;
```

```
print (QQ)

l = stationary * QQ; #np.identity(1,3);
# customers_per_time
lamda = np.sum(l);

l2 = -stationary*D0;
#batch_per_time
lg = np.sum(l2);
print (lg);
print (lamda);
```

Listing 5.4: Python code for computing arrival rate λ

3. We generate and save the BMAP matrices using the SpemFit tool in Matlab format to a file. We then launch it to Matlab and do the following:

```
I = ones(3);
Q = D0 + D1 + D2 + D3 + D4 + D5 + D6;

% compute stationary distribution - uniformalize Q
q=max(abs(diag(Q)));
P = (Q + q*eye(3)-diag(sum(Q')))/q;

pi0 = zeros(1,3);
pi0(1,1) = 1;
pi_n = pi0*P^1000000000000;
aux = 1*D1 + 2*D2 + 3*D3 + 4*D4 + 5*D5 + 6*D6;
lamda_mat = pi_n*aux*ones(3,1);
lg = -pi_n*D0*ones(3,1);
test = sum(pi_n);
```

Listing 5.5: Matlab code for computing arrival rate λ

Let us present below an example of the format of the specification of the arrival process in python:

```
import numpy.matlib as ml
D0 = ml.matrix([[-22.177, 0, 0], [0, -5.018, 0], [0, 0, -1488559.9215]])
D1 = ml.matrix([[14.24, 0.768, 1.401], [0.108,4.234, 0.319], [404474.388,
    705022.355, 290956.849]])
D2 = ml.matrix([[3.333, 0.404, 0.168], [0.025, 0.211, 3.953e-32], [52459.669,
     1.975e-74, 3630.994]])
D3 = ml.matrix([[1.099, 0.049, 0.153], [0.013, 0.083, 3.103e
    -47],[28704.722,0.006, 0]])
D4 = ml.matrix([[0.42, 0.021, 0.012], [0.007, 0.017, 1.042e-164], [3310.939,
    5.787e-96, 0]])
```

```
D5 = ml.matrix([[0.063, 0.008, 0.013], [3.085e-07, 0.003, 2.556e-98], [0, 0,
    0]])
D6 = ml.matrix([[0.026, 5.414e-63, 0], [2.13e-08, 3.93e-49, 0], [0, 0, 0]])
8.931
```

Listing 5.6: Arrival Fitting python file

service_fitting_file in python: In this file, we save the matrix of the representation of service process: the transition matrix A, and the vector of the initial probability α. In the last line of the file, we save the service rate. We use below SpemFit tool to generate the transition matrix A and the initial probability vector α. Let us present below an example of the format of service fitting filein python:

```
import numpy.matlib as ml
A = ml.matrix([[-10.0, 10.0, 0], [.4, -1.0, .1], [0, .1, -.1]])
alpha = ml.matrix([[.4, .5, .1]])
0.195312
```

Listing 5.7: Service Fitting python file

The service rate is denoted by $\mu = \frac{1}{\mathrm{E}[X]}$, where X is the random variable of service time. The arrival rate is saved to the last line of the service fitting file in python format.
experiment_title is part of the marmoteCore-Q output file names. More details related to the marmoteCore-Q output are presented in the following Section 5.1.5.

Remark: The user must respect strictly the python format of arrival and service files as presented above. Pay attention to new lines, spaces and missing or redundant bracket(s). Otherwise segmentation fault occur.

5.1.5 Output

The program output includes 3 files:

1. `experiment_title.txt`: In this file, the results of our model are saved. An example is presented below:

```
### Test_MM1K_as_map_medium_load2
### Date: Sun Jan 27 21:45:05 2019
# Number of servers: 1
# Buffer size: 4
# Total size: 5
# Number of arrival phases: 2
# Maximal batch size: 1
# Arrival matrices: Array(0..1, [ Matrix(2,2, [-0.4,0],0,-0.4]]), Matrix
    (2,2, [0.2,0],0,0.4]])]
# Number of service phases: 2
# Phase-type matrix: Matrix(2,2,[[-0.8,0],[0,-0.8]])
# Phase-type vector: Vector[row](2, [0.5,0.5])
# Reentry probability: 0
# Impatience rate: 0
```

```
### Solution method
# Power method
# Precision: 1e-07
### Results
#
# Index 0: queue size distribution
# Index 1: arrival phase distribution
# Index 2: busy servers distribution
# Index 3: service phase 0 distribution
# Index 4: service phase 1 distribution
#
### Agregate load statistics
# Customer arrival rate: 0.400000
# Customer service rate: 0.800000
# Offered load: 0.500000
### Computations of flows
# Accepted throughput: 0.399707
# Lost throughput: 0.000293
# Served throughput: 0.199853
# Reneged throughput: 0.000000
### Queue size statistics
# Average queue size: 0.331868
# Moment 2 queue size: 0.545786
# Variance queue size: 0.435650
### Performance Metrics
# Expected number of servers: 0.249817
# Expected waiting time: 0.205278
### Queue size distribution (n,p_n)
    0 0.750183
    1 0.187546
    2 0.046886
    3 0.011722
    4 0.002930
    5 0.000733
### Number of busy server distribution (c,p_c)
    0 0.750183
    1 0.249817


### Arrival phase distribution (f,p_f)
    1 1.000000
    2 0.000000
```

```
### Service phase distribution 0 (c,p_c)
    0 0.875092
    1 0.124908


### Service phase distribution 1 (c,p_c)
    0 0.875092
    1 0.124908
```

Listing 5.8: `experiment_title.txt`

2. `experiment_title.dat`: In this file, the stationary distribution is saved. An example is presented below:

```
discrete [ 0.0000 1.0000 2.0000 3.0000 4.0000 5.0000 6.0000 7.0000
    8.0000 9.0000 10.0000 11.0000 12.0000 13.0000 14.0000 15.0000 16.0000
     17.0000 18.0000 19.0000 20.0000 21.0000 ] [ 0.7502 0.0000 0.0938
    0.0000 0.0938 0.0000 0.0234 0.0000 0.0234 0.0000 0.0059 0.0000 0.0059
     0.0000 0.0015 0.0000 0.0015 0.0000 0.0004 0.0000 0.0004 0.0000 ]
```

Listing 5.9: `experiment_title.dat`

3. `experiment_title_maple.dat`: In this file, the stationary distribution is saved in maple format. An example is presented below:

```
Vector( [ 7.501833e-01,
   0.000000e+00,
   9.377288e-02,
   0.000000e+00,
   9.377288e-02,
   0.000000e+00,
   2.344319e-02,
   0.000000e+00,
   2.344319e-02,
   0.000000e+00,
   5.860783e-03,
   0.000000e+00,
   5.860783e-03,
   0.000000e+00,
   1.465189e-03,
   0.000000e+00,
   1.465189e-03,
   0.000000e+00,
   3.662953e-04,
   0.000000e+00,
   3.662953e-04,
   0.000000e+00
```

```
] );
```

Listing 5.10: `experiment_title_maple.dat`

In the experiment_title.txt file, we present the experiment input parameter, we then define indices for queue size, arrival phase, busy servers and service phases distributions. Indices help to plot the results using gnuplot.

5.2 Validation of the implementation

Complex models such that $BMAP/PH/N+c$ do not have analytical solutions, thus making it difficult to evaluate the results. To validate complex models, we present the following ways:

- We define and verify the conservation laws as described in Section 3.4.2.
- We check if the equations (3.20), (3.24) and (3.22) defined in Sections 3.4.5, 3.4.7 and 3.4.6 respectively are satisfied. The summation of service phase probabilities is equal to 1, as expected (see Section 3.4.8).
- We define and test three types of load. Let us first define the load factor below: For average arrival rate λ and service rate μ, the system utilization or load is defined by ρ:

$$\rho = \frac{\lambda}{c\mu} \tag{5.7}$$

  Load types are explained below:

  - **critical load**, when $\rho = 1$. Using critical load, we have simple mathematical formulas and we check everything at the same time.
  - **overload**, when $\rho > 1$. In overload case, the buffer size is large. Customers are more than system capacity (or overcome system capacity) and the system can not handle the load.
  - **underload**, when $\rho < 1$. Customers are fewer than the system capacity. Hence, system can handle the load. The system has a relatively small number of customers and, consequently, we expect that the probability that customers will be rejected is low. In other words, the loss probability should be zero or very small.

- To validate all the code parts, we first simulate simpler queueing system with known explicit closed functions and compare the results with the theoretical ones. Many cases are used for benchmarking.

Ideally, all of the above validation ways should be satisfied in order for the model results to be confirmed.

5.2.1 The modeler tricks

The exponential assumption is presented in most models for which an analytical solution is known. For this reason, we describe below how to emulate exponential distributions or Poisson proccess from $BMAP$ format for the arrival process and $Ph-type$ format for the service process. We then discuss the large buffer principle.

5.2.1.1 Service Process

For services, the Exponential distribution is achieved the following way:

- choose the matrix T as: $diag(-\mu, ..., -\mu)$
- pick any distribution β.

With such parameters, the service is exponential with parameter μ, and the probability that a given service in phase i is β_i. Different phase services are actually i.i.d with this distribution. Hence, in a C-server queue, the joint probability distribution of the number of servers in the different phases is given by:

$$P(\vec{s}) = p(|\vec{s}|)\binom{\vec{s}}{s_1, ..., s_c}\prod_{j=1}^{S}\beta_j^{s_j} \tag{5.8}$$

5.2.1.2 Arrival Process

For arrivals, we choose $D_1 = \lambda P$ and $D_0 = -\lambda I$, where P is any ergodic probability transition matrix. In that case, the arrival process is Poisson with rate λ and the stationary distribution of the arrival phase is the stationary distribution of P.

5.2.1.3 The large buffer principle

The usual behavior of an exponential queueing system is that the probability distribution of the number of customers decreases exponentially as a function of this number. As a consequence, as the size N of the buffer grows, the distribution of the queue with finite buffer N and that of the same queue with infinite buffer ∞ get closer and closer. In practice, we can still use the marmoteCore-Q for the infinite cases, using the formulas of queueing theory for $N = \infty$ as an approximation of the numerical results for N finite but large.

We show below some numerical cases and evaluate our results against theoretical ones, which are based on the closed-formulas in the literature. We implement the well-known closed formulas in maple and compare them with marmoteCore-Q results. For more details on these formulas the reader can refer to AppendixF. Maple output files have the same format as we present in Section 5.1.5 so that they are comparable with a simple diff command.

5.2.2 Numerical test-cases

Using the above "tricks" we test the following cases and give an example for each one:

1. Basic markov queues $M/M/1/K$ and $M/M/1$. An example of $M/M/1$ with high load ($\mu > \lambda$) is described below. The input parameters are:
   - Poisson process with arrival rate with $\lambda = 2.5$, the maximum number of batch size is $K = 1$ and D_0, D_1 are scalars as described in Section 5.2.1.2. Hence, for $\lambda = 2.5$ we have: $D_0 = -2.5$ and $D_1 = 2.5$
   - Poisson process with service rate $\mu = 3$ and the matrix T and the vector β are scalars since there is only one phase. Hence, for $\mu = 3$ we have: $T = -3$ and $\beta = 1$

- buffer size is very large since the system capacity is infinite. For instance, $N = 10000$.

Some of the output results are presented below:

```
### Agregate load statistics
# Customer arrival rate: 2.500000
# Customer service rate: 3.000000
# Offered load: 0.833333
### Computations of flows
# Accepted throughput: 2.500000
# Lost throughput: 0.000000
# Served throughput: 2.500000
# Reneged throughput: 0.000000
### Number of busy server distribution (c,p_c)
    0 0.166667
    1 0.833333

### Arrival phase distribution (f,p_f)
    1 1.000000

### Service phase distribution 0 (c,p_c)
    0 0.166667
    1 0.833333
```

Observe that the results satisfy the equations (3.22), (3.2), (3.3), (3.20) and (3.4). The summation of service phase probabilities is equal to 1, as expected (see Section 3.4.8). In this case, even the system is underload (offered load = 0.833333<1) and the lost throughput is zero as expected. Finally, we compare the empirical results with the ones of theoretical closed-form formulas and verify that they match. For more details on theoretical closed-form formulas, the reader can refer to Appendix F1.

2. Multi-server, impatience and reentering: $M/M/C/N$ with and without impatience and reentry. In this category, we have many servers and customers that may abandon the system and others that try again to be served. We analyze below $M/M/C/N$ with impatience. The input parameters are presented below:

   - Poisson arrival process can be expressed as Markovian Arrival Process (MAP). In this case, maximum batch size $K = 1$ and maximum number of arrival phases $A > 1$. Hence, the matrices D_0 and D_1 for $\lambda = 4$ are defined as follows:
   $$D_0 = \begin{bmatrix} -4 & 0 & 0 \\ 0 & -4 & 0 \\ 0 & 0 & -4 \end{bmatrix} \text{ and } D_1 = \begin{bmatrix} 2 & 0 & 2 \\ 0 & 0 & 4 \\ 2 & 2 & 0 \end{bmatrix}$$
   - Exponential service time distribution can be expressed as follows:

$$T = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \text{ and } \beta = \begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 \end{bmatrix}$$

- buffer size = 8, number of servers = 4 and impatience rate = 1.

Some of the output results are demonstrated below:

```
### Agregate load statistics
# Customer arrival rate: 4.000000
# Customer service rate: 1.000000
# Offered load: 1.000000
### Computations of flows
# Accepted throughput: 3.997433
# Lost throughput: 0.002567
# Served throughput: 3.218319
# Reneged throughput: 0.779114

### Number of busy server distribution (c,p_c)
    0 0.018321
    1 0.073283
    2 0.146565
    3 0.195420
    4 0.566411

### Arrival phase distribution (f,p_f)
    1 0.400000
    2 0.200000
    3 0.400000

### Service phase distribution 0 (c,p_c)
    0 0.676996
    1 0.270712
    2 0.049727
    3 0.002508
    4 0.000057

### Service phase distribution 1 (c,p_c)
    0 0.523755
    1 0.325491
    2 0.127687
    3 0.022278
    4 0.000789
```

Observe that the system is at critical load ($\rho = 1$) but the lost throughput is larger than zero. In this case, customers are impatience with rate 1. The equations (3.2), (3.3) and,

(3.4), (3.20), (3.24) are satisfied. The summation of each service phase probabilities is equal to 1, as expected. For more details on theoretical closed-form formulas for $M/M/C/N$, the reader can refer to Appendix F2.

3. Phase-type service: $M/PH/1$ and $M/PH/\infty$. In order to simulate the queueing model $M/PH/\infty$, we need to give to marmoteCore-Q tool a finite number of servers and buffer capacity. An example for $M/PH/\infty$ can be $M/PH/50/1000$ with number of servers $c = 50$ and buffer size $N = 1000$ and is demonstrated below:

   - Arrival matrices: $D_0 = -1$ and $D_1 = 1$
   - Phase-type matrix & phase-type vector: $T = \begin{bmatrix} -10 & 10 & 0 \\ 0.4 & -1 & 0.1 \\ 0 & 0.1 & -0.1 \end{bmatrix}$ and $\beta = \begin{bmatrix} 0.4 & 0.5 & 0.1 \end{bmatrix}$.

   Some of the output results are demonstrated below:

```
### Agregate load statistics
# Customer arrival rate: 1.000000
# Customer service rate: 0.195312
# Offered load: 0.102400
### Computations of flows
# Accepted throughput: 1.000000
# Lost throughput: 0.000000
# Served throughput: 0.999873
# Reneged throughput: 0.000000
### Arrival phase distribution (f,p_f)
    1 1.000000
```

   Observe that the equations (3.2), (3.3) and (3.4) are not fully satisfied. However, the difference is 127×10^{-6} which can be acceptable, given that the stationary distribution is approximately computed. Another explanation could be that we cannot pass as an input to the program infinite number of servers and buffer size. Hence, for models with infinite number of servers and/or buffer size, it is expected to have numerical errors in scale of 10^{-6}. Finally, comparing the empirical results with the ones of theoretical closed-form formulas and we conclude that the results are satisfactory.

4. Batch arrivals: $M_X/M/1$ and $M_X/M/\infty$ An example with 2 batches and number of servers $c = 1$ is discussed below. The input parameters are defined as follows:

   - Arrival process is a Poisson process with 2 batches and arrival rate $\lambda = 0.4$ and the matrices are: $D_0 = -0.4$, $D_1 = 0.2$ and $D_2 = 0.2$
   - Poisson service process with service rate $\mu = 0.8$
   - buffer size $N = 1000$

   Some of the output results are presented below:

```
### Agregate load statistics
# Customer arrival rate: 0.600000
# Customer service rate: 0.800000
# Offered load: 0.750000
### Computations of flows
# Accepted throughput: 0.600000
# Lost throughput: 0.000000
# Served throughput: 0.600000
# Reneged throughput: 0.000000
### Number of busy server distribution (c,p_c)
    0 0.250000
    1 0.750000

### Arrival phase distribution (f,p_f)
    1 1.000000

### Service phase distribution 0 (c,p_c)
    0 0.250000
    1 0.750000
```

Observe that the results satisfy the equations (3.22), (3.2), (3.3), (3.20) and (3.4). The summation of service phase probabilities is equal to 1, as expected. For more details on theoretical closed-form formulas for $M_X/M/1$, the reader can refer to Appendix F3.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this dissertation, we presented new models for solar power and real data center clusters. For the last purpose, two different types of data center clusters workload were characterized in detail and a tool was proposed, which simulates both simple queueing models, with known analytical solutions, and complex ones.

Our initial concern was enabling data centers to consume all their electricity from renewable energy sources e.g. solar, wind and water. An obvious challenge associated with renewable energy sources is the unreliability of supply. Renewable energy often relies on the weather for its source of power. For instance, hydrogenerators need rain to fill dams for flowing water supply, wind turbines need wind to turn their blades and solar collectors need clear skies and sunshine to collect heat and produce electricity. When these resources are unavailable so is the potential to generate energy from them. This can be unpredictable and inconsistent. Consequently, a promising research direction is to construct models that accurately predict the energy supply subject to the weather conditions. In this thesis, we focused on solar renewable energy which is clean, safe and cheap and is abundant in many locations. We developed a stochastic model that models the solar irradiance at the surface of the Earth and a second one that, given the photovoltaic output, models the photovoltaic output current. We then compared both models with the on-off power source model developed by Miozzo et al. [Miozzo 2014] and concluded that our solar model at the surface of the Earth outperformed the other models. We finally validated both models with real solar irradiance and we concluded that modeling the solar irradiance is more efficient and gives more accurate results compared to the actual solar traces. Our model captures the multiscale correlations that are inherently present in the solar irradiance and the time-scales that are reproduced by the model performs well. We finally concluded that our stochastic model is suitable for small scale cases and can be used in modern domains such as ITC applications.

The other issue that we addressed in this thesis was the capacity planning provision for data center clusters. A major factor which plays an important role in making the capacity planning is the data center cluster's workload. Based on the data center cluster clients' demands for both data space and reliable access to their data, the data center cluster should adapt its capacity. Hence, a model for data center cluster workload forecasting is necessary. We studied that, in such systems, executed jobs are submitted by users and these jobs may consist of more than one task. Some jobs may abandon the system before starting to be served and some others may be executed more than once. A first step towards capturing the above features is to propose a model that can potentially predict the data center clusters' workload. This model overcomes the literature stereotypes related to the exponential assumptions for arrival and service process since it does not correspond to the real ones. It is a

multi-server queueing model with the arrival process determined as Batch Markovian Arrival Process (BMAP) and the service times fitted to a phase-type distribution. Impatient clients' abandonments and users' resubmissions are taken into account as well. To our knowledge, this model has not been analyzed in the literature. We conclude that our queueing model for real data center clusters captures the features of real data center clusters.

To support our model assumptions, we analyzed current capacity, characterizing two different types of workloads; the Google one which serves Google employees (or Google internal needs) and the scientific one which is collected by Nef cluster, the data center cluster of research center of French National Institute for Computer Science and Automation (Inria) in Sophia Antipolis. We present the characteristics of the traces and the statistics on task level. We then characterized, in both traces, the arrival and service process, the abandonments and waiting times. We concluded that in both cases, the abandonment process can fit to a Burr distribution, with a Weibull and a exponential distribution to give us acceptable results. It is worth pointing out that the customers wait less on average in the Google cluster in comparison to the Nef cluster. Although this could be explained by the fact that the machines in the two clusters differ in terms of computing power and in terms of hardware, this factor is hardly enough to account for such a high difference in the number of machines that correspond to the user; the number of machines per user is around 13 in the Google traces case, whereas in the Nef traces case, it is around 0.5 machines per user, namely twenty six times less than in the Google traces case. As far as the service time is concerned, we showed that, in both cases, the service time can be fitted to a phase-type distribution. We also conclude that the execution type has an impact on the service time. In the Google cluster case, the minimum average service time is for when jobs failed, whereas in the Nef cluster case, the minimum one is for the jobs which have been successfully executed. Moreover, in both traces, we observed correlations in the interarrival times and different range of batch sizes. Hence, we showed that a Batch Markov Arrival Process can be considered to capture the correlations observed in real workload submissions.

Another issue concerns the model's extraction parameters which will be used for its simulation. In such systems, the large number of servers is a very important problem and influences the problem complexity. Essentially, the smaller the number of system servers the better. Another conclusion that can be reached is that in order to overcome the technical challenges of the simulation we needed to find subclusters that can be representative of the whole clusters in order to simulate our model.

Finally, we believe that the scientific queueing community needed a tool for queueing simulations, even for complex queues, which does not make use of explicit closed-formulas and which gives correct and accurate results in a faster way. For this reason, we developed the marmoteCore-Q tool, using the marmoteCore platform[Jean-Marie 2017]. We used this tool to validate our BMAP/PH/c model with abandonments and resubmissions for data center clusters. Moreover, with the marmoteCore-Q tool one can simulate queues with known analytical results such as the $M/M/1$ queue and its finite capacity version, the $M/M/c/K$ queue, the $M/PH/1$ and $M/PH/\infty$ queues, the $M^X/M/1$ and $M^X/M/\infty$ queues, etc.

6.2 Future Work

Solar Power Perspectives.

Concerning the solar power part of this thesis, we proposed a 4-state semi-Markov model for solar irradiance for the city of Los Angeles. A perspective can be to investigate whether the same 4-state semi-Markov model can be tuned and validated using data related to another city. Each state in this model refers to a given weather condition. Obviously, the empirical distributions of sojourn times and values in each state from the retrieved traces will be changed for different regions. The questions if this model can be universal and how much the same geographical morphology even in the same range of latitude and longtitute can change the results are still open. Another open question is in which region is better to locate a data center cluster. This question can be replied after comparing the results of the most sunny regions and estimate how much energy we have per photovoltaic panel in short time scale.

Data center cluster Perspectives.

Regarding the queueing model for data center clusters, we saw that, for simplicity, as a first step we chose to fit the abandonment process with the exponential distribution, even though the best fit was the Burr one. Hence, a next step could be to include the abandonment as Burr distribution in our model and propose numerical analytical solutions for the impatient process. Extended this work by simulating our queueing model, using marmoteCore-Q and evaluate the results.

Another direction that could be explored is the development of a workload generator for real data center clusters based on our proposed queueing model. Research devoted to capacity provision, sheduling problems, cooling, etc needs actual data center cluster traces to be validated. Providing traces to the researchers is a very complicated and difficult task due to the privacy and security issues which have been prevalent during the past years all around the world. Therefore, being able to generate real workload for a data center would be extremely valuable to the scientific data center community.

Another perspective is the proposition of a fluid model for the real data center cluster. In reality, the arrival process in real data center clusters shows non-stationarity, since the system does not empty. The only time that the system is empty is when the data center cluster is launched for the first time or after maintenance, in which case all the servers are turned off for a given period. The advantage of the fluid models is that they are not sensitive to the arrival phase approximation and, thus, there is no need to assume stationarity.

Another perspective is the proposition of a data center simulator that integrates precise energy provisioning and distribution models when connected to multiple sources (e.g. a grid and a local PV array). Despite the fact that data center simulators exist (see Simgrid [Casanova 2014]), their power models are related to the server and network consumption, not the provisioning and the distribution aspects.

In real data center clusters, the phenomena of processor-sharing has been observed in practice. Thus, queueing models such as Processor-Sharing (PS) scheduling policies can be proposed to model real data center clusters and their energy consumption of processors.

As far as the workload characterization is concerned, we need to parameterize both Google and Nef clusters. Thus, open questions like the maximum number of machines and the maximum number of tasks served at the same time that permits us to overcome the technical challenges and simulate our model, needs more investigation. Furthermore, as for the Nef cluster, we are going to extract representative subclusters that can be used to our future numerical analysis, as discussed in Section 4.8.

Combining our solar power model and our queueing model for real data center clusters, it can be helpful to answer open research questions pertaining to the optimisation of the data center's operations. Questions such as optimising the capacity planning decisions, workload scheduling and finding the optimal buying price for electricity according to demands and geographical location. These models can be extended to solve different problems regarding other data center subsystems. In particular, there are open research issues about improving equipment or the cooling infrastructure.

Finally, open research topics related to the green data center clusters can be if the schedulers compatible with the intermittent nature of a photovoltaic power sources, what are the theoretical bounds that prevent to power an existing datacenter using photovoltaic arrays, coupled or not with power storage systems, how to size a photovoltaic array on top of an existing data center depending on the targeted environmental and economic objectives.

Appendix

A k-means Clustering Algorithm

A1 Description

Given a set of observations (x_1, x_2, ..., x_n), where each observation is a d-dimensional real vector, k-means clustering aims to partition the n observations into k sets $S = \{S_1, S_2, ..., S_k\}$ so as to minimize the within-cluster sum of squares:

$$\sum_{i=0}^{n} \min_{\mu_j \in S} (||x_i - \mu_j||^2) \tag{A1}$$

where μ_j is the mean of the samples in the cluster and so-called "centroids".

A2 The Algorithm

The general idea about k-means algorithm is presented below:

Algorithm 2 k-means algorithm

Input: Dataset
1. Place k points into the space represented by the objects that are being clustered. These points represent initial group centroids.
2. Assign each object to the group that has the closest centroid.
3. When all objects have been assigned, recalculate the positions of the k centroids.
4. Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

Result: optimal k, k-clusters

A3 Davies-Bouldin Index

The Davies-Bouldin Index is a metric to define the optimal k. The Davies-Bouldin criterion is based on a ratio of within-cluster and between cluster distances. It is defined as:

$$DB = \frac{1}{k} * \sum_{i=1}^{k} \max_{j \neq i} D_{i,j} \tag{A2}$$

where, $D_{i,j} = \frac{\bar{d}_i + \bar{d}_j}{d_{i,j}}$. $\bar{d}_i$ is the mean distance between the centroid of the ith cluster and each point in the ith cluster. $\bar{d}_j$ is the mean distance between the centroid of the jth cluster and each point in the jth cluster. $d_{i,j}$ is the Euclidean distance between the jth and ith centroids of corresponding clusters.

The smallest Davies-Doublin index value ($D_{i,j}$) represents the optimal clustering solution and the maximum value the worst one.

B Autocorrelation Function

Suppose that y_t is a stochastic process, the autocorrelation function (ACF) measures the correlation between y_t and y_{t+k}, where $k = [0, ..., K]$, $k \in \mathbb{Z}$.

The autocorrelation for lag k is:

$$r_k = \frac{c_k}{c_0}, \tag{B1}$$

where

- $c_k = \frac{1}{T}\sum_{t=1}^{T-k}(y_t - \bar{y})(y_{t+k} - \bar{y})$
- c_0 is the sample variance of the time series.

The estimated standard error (SE) of the autocorrelation at lag $k > q$, where q is the lag is beyond which the theoretical ACF is effectively 0, defined as:

$$SE(r_k) = \sqrt{\frac{1}{T}(1 + 2\sum_{j=1}^{q} r_j^2)}. \tag{B2}$$

The standard error reduces to $\frac{1}{\sqrt{T}}$ in case of completely random series.

C Periodogram

The periodogram is a nonparametric estimate of the power spectral density (PSD) of a wide-sense stationary random process. The periodogram is the Fourier transform of the biased estimate of the autocorrelation sequence. For a signal, x_n with sample rate f_s [1], the periodogram is defined as:

$$\hat{P(f)} = \frac{\Delta t}{N}|\sum_{n=0}^{N-1} x_n e^{-i2\pi f_n}|^2, -1/2\Delta t < f \leq 1/2\Delta t \tag{C1}$$

where Δt is the sampling interval. For a one-sided periodogram, the values at all frequencies except 0 and the Nyquist, $1/2\Delta t$, are multiplied by 2 so that the total power is conserved. For frequencies which are in sample, the periodogram is defined as:

$$\hat{P\omega} = \frac{1}{2\pi N}|\sum_{n=0}^{N-1} x_n e^{-i\omega n}|^2, -\pi < \omega \leq \pi. \tag{C2}$$

The integral of the true PSD, $P(f)$, over one period, $1/\Delta t$ for cyclical frequency, is equal to the variance of the wide-sense stationary random process:

$$\sigma^2 = \int_{\frac{-1}{\Delta t}}^{\frac{-1}{\Delta t}} P(f)df. \tag{C3}$$

[1] The sample rate is the number of samples per unit time

For normalized frequency 2π, the $P(f)$ over one period is:

$$\sigma^2 = \int_{-2\pi}^{2\pi} P(f)df. \tag{C4}$$

D Burr distribution

The Burr Type XII distribution or simply the Burr distribution [Burr 1942] is a continuous probability distribution for a non-negative random variable.

The Burr distribution has probability density function:

$$f(x; c, k) = ck\frac{x^{c-1}}{(1+x^c)^{k+1}}$$

and the cumulative distribution function:

$$F(x; c, k) = 1 - (1+x^c)^{-k}$$

The complementary cumulative distribution function is defined as:

$$\bar{F}(x; c, k) = 1 - F(x) = (1+x^c)^{-k}.$$

For $c = 1$, the Burr distribution becomes the Pareto Type II (Lomax) [2]. The Burr distribution has mean:

$$\mu_1 = kB(\frac{k-1}{c}, 1+\frac{1}{c}),$$

where $B()$ is the beta function [3] and variance:

$$Var(X) = \mu_1^2 + \mu_2$$

where μ_2 is the second moment of random variable X.

E marmoteCore-Q Installation Steps

We present below the marmoteCore-Q installatin steps:

1. Download and install marmoteCore from here
2. Replace the Makefile, which is provided by marmoteCore with the one in Listing 6.1
3. To compile the marmoteCore-Q, following the same instructions as presented in here. Specifically,
   - If marmoteCore was installed globally in directory MAR_DIR, do:

[2] The Pareto Type II (Lomax) has complementary cumulative distribution $\bar{F}(x) = \left(1+\frac{x}{\sigma}\right)^{\alpha}$, $x \geq 0$ and $\sigma > 0, \alpha$

[3] The beta function is defined by $B(x, y) = \int_0^1 t^{x-1}(1-t)^{y-1}dt$.

```
make MARMOTEDIR=MAR_DIR
```

- If marmoteCore was installed locally, do:

```
make Local=true MARMOTEDIR=MAR_DIR
```

4. Run the command `./marmoteCore-Q` with the appropriate parameters as discussed in Section 5.1.4.

```
# Choice of Marmote library: distrib/installed or development
ifdef DISTRIB
MARMOTEDIR=/usr/local/marmotecore
INCLUDEDIR=$(MARMOTEDIR)/include
else ifdef LOCAL
INCLUDEDIR=$(MARMOTEDIR)/include
else
MARMOTEDIR=../../../marmotecoredev
INCLUDEDIR=$(MARMOTEDIR)
endif
LIBDIR=$(MARMOTEDIR)/lib

# Flag for interaction with R
ifdef WITH_R
RFLAGS = -DWITH_R -I/usr/include/R -I/usr/lib64/R/library/Rcpp/include -I/usr
    /lib64/R/library/RInside/include -L/usr/lib64/R/lib -lR -L/usr/lib64/R/
    library/RInside/lib -lRInside -Wl,-rpath,/usr/lib64/R/library/RInside/lib
else
RFLAGS =
endif

# Choice of C compiler and options
ifeq ($(OS),Windows_NT)
CFLAGS += -std=gnu++11
endif

CPPCOMPILER = g++

CFLAGS += -std=gnu++11 -Wall -pedantic -g
VAL=valgrind --leak-check=full --show-leak-kinds=all

LIBRARIES=$(addprefix -l, MarmoteCore Xborne psi boost_thread
    boost_filesystem)
APPLIS=bmap_f

all: $(APPLIS)
```

```
%: %.cpp
	$(CPPCOMPILER) $(CFLAGS) -I$(INCLUDEDIR) $^ -o $@ -L$(LIBDIR) $(
	    LIBRARIES)

clean:
	/bin/rm $(APPLIS)
```

Listing 6.1: Makefile

F Queues

F1 $M/M/1$ queue

When services are exponential, there is a unique phase. However, this unique phase can be represented with a phase space of arbitrary size, see Section 5.2.1.1. In the case where the representation is with a single phase (S = 1), the vector describing services is either $\vec{s} = 0$ (empty server) or $\vec{s} = 1$ (busy server).

Using Pollaczek–Khinchine (PK) [4] formula, the probability generating function of N is:

$$\Pi(z) = \frac{1-\rho}{1-\rho^z} \tag{F1}$$

In the following, the average queue size (including service), its second moment and its variance for $M/M/1$ are defined:

$$\mathbb{E}[N] = \frac{\rho}{1-\rho} \tag{F2}$$

$$\mathbb{E}[N^2] = \frac{\rho}{1-\rho} + \frac{2\rho^2}{(1-\rho)^2} \tag{F3}$$

$$Var[N] = \frac{\rho}{(1-\rho)^2} \tag{F4}$$

The queue size distribution is known:

$$\pi_n = (1-\rho)\rho^n, \text{ for every } n \in N \tag{F5}$$

[4] Pollaczek-Khinchine formula is $L = \rho + \frac{\rho^2 + \lambda^2 * Var(S)}{2(1-\rho)}$, where λ is the arrival Poisson process, $\frac{1}{\mu}$ is the mean of the service time distribution S, $\rho = \frac{\lambda}{\mu}$ is the utilization, $Var(S)$ is the variance of the service time distribution S.

Using Little's Law [5], the mean time in the system is:

$$\mathbb{E}[T] = \frac{\mathbb{E}[N]}{\lambda} = \frac{1}{\mu - \lambda} \tag{F6}$$

Next, the expected waiting time is:

$$\mathbb{E}[W] = \mathbb{E}[T] - \frac{1}{\mu} = \frac{\rho}{\mu - \lambda} \tag{F7}$$

F2 $M/M/C/N$ queue

As we mentioned in Section F1, when services and arrivals are exponential, there are a unique phase for phase-type distribution and batch markovian arrival process (see Section 5.2.1.2) respectively. Hence, the system can be modeled as a birth-death process using the following respective arrival and service time:

$$\lambda_n = \lambda, \text{ if } 0 \leq n \leq N-1 \tag{F8}$$

$$\mu_n = \begin{cases} n\mu, & \text{if } 0 \leq n \leq c \\ c\mu, & \text{if } c \leq n \leq N \end{cases} \tag{F9}$$

The stationary distribution can be written as:

$$\pi_n = \begin{cases} \frac{\alpha^n}{n!} p_0, & \text{if } 0 \leq n \leq c \\ \frac{\alpha^n}{c^{n-c} c!} p_0, & \text{if } c \leq n \leq N \end{cases} \tag{F10}$$

where $\alpha := \frac{\lambda}{\mu}$ and

$$\pi_0 = \begin{cases} (\sum_{n=0}^{c-1} \frac{a^n}{n!} + \frac{a^c}{c!} \frac{1-\rho^{N-c+1}}{1-\rho})^{-1}, & \text{if } \rho \neq 1 \\ (\sum_{n=0}^{c-1} \frac{\alpha^n}{n!} + \frac{\alpha^c}{c!}(N-c+1))^{-1}, & \text{if } \rho = 1 \end{cases} \tag{F11}$$

The mean queue length is:

$$\begin{aligned} \mathbb{E}[Q] &= \sum_{n=c+1}^{N} (n-c)\pi_n \\ &= \sum_{n=c+1}^{N} (n-c) \frac{\alpha^n}{c^{n-c} c!} \pi_0 \end{aligned} \tag{F12}$$

[5] Little's Law is defined as: $L = \lambda * W$, where L is the average number of customers in a stationary system, λ is the arrival rate and W is the average time that a customer spends in the system.

$$
\begin{aligned}
&= \frac{\pi_0 \rho_c}{c!} \sum_{n=c+1}^{K} (n-c) \frac{\rho^{n-c}}{c^{n-c}} \\
&= \frac{\pi_0 \rho^c \alpha}{c!} \sum_{n=c+1}^{N} (n-c) \alpha^{n-c-1} \\
&= \frac{\pi_0 \rho^c \alpha}{c!} \sum_{i=1}^{K-c} i \alpha^{i-1} \\
&= \frac{\pi_0 \rho^c \alpha}{c!} \frac{\mathrm{d}}{\mathrm{d}\alpha} \Big(\sum_{i=0}^{N-c} \alpha^i \Big) \\
&= \frac{\pi_0 \rho^c \alpha}{c!} \Big(\frac{1-\alpha^{N-c+1}}{1-\alpha} \Big)
\end{aligned}
$$

$$
\mathbb{E}[Q] = \frac{\pi_0 \rho^c \alpha}{c!(1-a)^2} [1 - \alpha^{(N-c+1)} - (1-a)(N-c+1)\alpha^{(N-c)}] \tag{F13}
$$

The average number of customers in the system is:

$$
\mathbb{E}[N] = \frac{\lambda}{\mu} + \pi_0 \frac{\rho (c\rho)^c}{(1-\rho)^2 c!} \tag{F14}
$$

The mean time in the system is:

$$
\mathbb{E}[T] = \frac{1}{\mu} + \pi_0 \frac{\rho (c\rho)^c}{(1-\rho)^2 c!} \tag{F15}
$$

Next, the expected waiting time is:

$$
\mathbb{E}[W] = \frac{\mathbb{E}[Q]}{\lambda(1-\pi_N)} = \frac{\pi_0 \rho^c \alpha [1 - \alpha^{(N-c+1)} - (1-\alpha)(N-c+1)\alpha^{N-c}]}{\lambda \Big(1 - \frac{\alpha^N \pi_0}{c^{N-c} c!} \Big)} \tag{F16}
$$

F3 $M_X/M/1$ queue

In $M_X/M/1$ queue, the customers arrive in batches in a Poisson process with parameter λ. The balance equations of the $M_X/M/1$ queue write as:

$$
(\lambda + \mu)\pi_n = \lambda \sum_{k=0}^{n-1} \pi_k p_{n-k} + \mu \pi_{n+1}, n \leq 1 \tag{F17}
$$

$$
\lambda \pi_0 = \mu \pi_1 \tag{F18}
$$

Observe that with the convention $\pi_0 = 0$, we can write $\sum_{k=1}^{n} \pi_k p_{n-k} = \sum_{k=0}^{n} \pi_k p_{n-k}$. Let $\Pi(z) = \mathbb{E}(z^N) = \sum_{n=0}^{\infty} \pi_n z^n$. From the balance equations we obtain:

$$
(\lambda + \mu) * (\Pi(z) - \pi_0) = \lambda * \sum_{n=1}^{\infty} \sum_{k=0}^{n} \pi_k p_{n-k} z^n + \mu * \frac{1}{z} (\Pi(z) - \pi_0 - z\pi_1)
$$

$$
\begin{aligned}
(\lambda+\mu)\Pi(z) - \mu * \pi_0 &= \lambda\Pi(z)X*(z) + \mu * \frac{1}{z}(\Pi(z)-\pi_0) \\
\Pi(z)(\lambda+\mu-\lambda X*(z)-\frac{\mu}{z}) &= \mu\pi_0(1-\frac{1}{z}) \\
\Pi(z) &= \frac{\mu\pi_0(z-1)}{\lambda z(1-X*(z))+\mu(z-1)}
\end{aligned}
\tag{F19}
$$

We determine π_0 by letting $z \to 1$. With L'Hôpital rule, we get $\pi_0 = 1 - \lambda\mathbb{E}[\frac{X}{\mu}] = 1-\rho$ with the definition $\rho := \lambda\mathbb{E}\frac{X}{\mu}$. Then the problem solution can be expressed as:

$$
\Pi(z) = \frac{1-\rho}{1-\frac{\rho^z}{\mathbb{E}[x]}\frac{1-X*(z)}{1-z}} \tag{F20}
$$

. In this expression, the function

$$
\overrightarrow{X}(z) := \frac{z}{\mathbb{E}X}\frac{1-X*(z)}{1-z} = \sum_{k=1}^{\infty}\frac{\mathbb{P}(X\le k)}{\mathbb{E}(X)}z^k \tag{F21}
$$

is a probability generating function: that of the "residual batch size".

The queue size moments are then obtained by differentiation:

$$
\mathbb{E}[N] = \frac{\rho}{2}\frac{\mathbb{E}[X^2]+\mathbb{E}[X]}{\mathbb{E}[X](1-\rho)} \tag{F22}
$$

Bibliography

[a] *BuTools V2.0 package, with Matlab, Mathematica and Python/NumPy support including the documentation, Url = http://rredc.nrel.gov/solar/old_ data/nsrdb/1991-2010/.*

[b] *Marmote: Download and installation instructions, Url = http://marmotecore.gforge.inria.fr/dokuwiki/doku.php?id=dowloadinstall.*

[c] *NEF documentation, Url = https://wiki.inria.fr/ClustersSophia/Clusters_ Home/.*

[d] *OAR 2.5 documentation, Url = http://oar.imag.fr/docs/2.5/.*

[e] *BuTools, Url = http://webspn.hit.bme.hu/ telek/tools/butools/index.php?page=6/.*

[f] *Solar Resource & Meteorological Assessment Project (SOLRMAP), Southwest Solar Research Park (Formerly SolarCAT).*

[g] *SPEMFIT, Url = https://bitbucket.org/ghorvath78/spemfit/.*

[Adaptive Computing 2018] Inc Adaptive Computing. *TORQUE-Adaptive Computing*, 2018.

[Amvrosiadis 2017] George Amvrosiadis, Jun Woo Park, Gregory R. Ganger, Garth A. Gibson, Elisabeth Baseman and Nathan DeBardeleben. *Bigger, Longer, Fewer: what do cluster jobs look like outside Google?* Technical report CMU-PDL-17-104, Parallel Data Laboratory, Carnegie Mellon University, 2017.

[Amvrosiadis 2018] George Amvrosiadis, Jun Woo Park, Gregory R. Ganger, Garth A. Gibson, Elisabeth Baseman and Nathan DeBardeleben. *On the diversity of cluster workloads and its impact on research results.* In 2018 USENIX Annual Technical Conference (USENIX ATC 18), Boston, MA, 2018. USENIX Association.

[Andrae 2015] Anders S.G. Andrae and Tomas Edler. *On Global Electricity Usage of Communication Technology: Trends to 2030.* Challenges, no. 6, pages 117–157, 2015.

[Andreas 2012] A. Andreas and S. Wilcox. *Solar Resource & Meteorological Assessment Project (SOLRMAP): Rotating Shadowband Radiometer (RSR); Los Angeles, California (Data).* Report DA-5500-56502, NREL, 2012.

[Artis 1979] H. Pat Artis. *Capacity Planning for MVS Computer Systems.* SIGMETRICS Perform. Eval. Rev., vol. 8, no. 4, pages 45–62, December 1979.

[Baba 1983] Yutaka Baba. *An algorithmic solution to the M/PH/c queue with batch arrivals.* J. Oper. Res. Jpn, vol. 26, no. 1, pages 33–50, March 1983.

[Barford 1998] Paul Barford and Mark Crovella. *Generating representative Web workloads for network and server performance evaluation.* vol. 26, no. 1, pages 151–160, June 1998.

[Bird 1981] Richard E. Bird and Roland L. Hulstrom. *A Simplified Clear Sky Model for Direct and Diffuse Insolation on Horizontal Surfaces.* Technical report Technical report SERI/TR-642-761, Solar Energy Research Institute, February 1981.

[Breuer 2001] Lothar Breuer. *The Periodic BMAP/PH/c Queue.* Queueing Systems, vol. 38, no. 1, pages 67–76, May 2001.

[Breuer 2002a] Lothar Breuer. *An EM Algorithm for Batch Markovian Arrival Processes and its Comparison to a Simpler Estimation Procedure.* Annals of Operations Research, vol. 112, no. 1, pages 123–138, Apr 2002.

[Breuer 2002b] Lothar Breuer, Alexander Dudin and Valentina Klimenok. *A Retrial BMAP/PH/N System.* Queueing Systems, vol. 40, no. 4, pages 433–457, May 2002.

[Burr 1942] Irving W. Burr. *Cumulative Frequency Functions.* The Annals of Mathematical Statistics, vol. 13, no. 2, pages 215–232, 1942.

[Capit 2005] N. Capit, G. Da Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounie, P. Neyron and O. Richard. *A batch scheduler with high level components.* In CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005., volume 2, pages 776–783 Vol. 2, May 2005.

[Casanova 2014] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson and Frédéric Suter. *Versatile, scalable, and accurate simulation of distributed applications and platforms.* Journal of Parallel and Distributed Computing, vol. 74, no. 10, pages 2899 – 2917, 2014.

[Chakravarthy 2002] Srinivas R. Chakravarthy and Alexander N. Dudin. *A Multi-Server Retrial Queue with BMAP Arrivals and Group Services.* Queueing Systems, vol. 42, no. 1, pages 5–31, Sep 2002.

[Cover 1991] Thomas M. Cover and Joy A. Thomas. Elements of information theory. Wiley-Interscience, New York, NY, USA, 1991.

[D. 2011] Cordeiro James D. and Kharoufeh Jeffrey P. Batch markovian arrival processes (bmap). American Cancer Society, 2011.

[Dave 1975] J. V. Dave, P. Halpern and H. J. Myers. *Computation of Incident Solar Energy.* IBM Journal of Research and Development, vol. 19, no. 6, pages 539–549, November 1975.

[Dayar 2016] Tugrul Dayar and M. Can Orhan. *Steady-state analysis of a multiclass MAP/PH/c queue with acyclic PH retrials.* Journal of Applied Probability, vol. 53, no. 4, page 1098–1110, 2016.

[Di 2012a] S. Di, D. Kondo and W. Cirne. *Characterization and Comparison of Cloud versus Grid Workloads.* In 2012 IEEE International Conference on Cluster Computing, pages 230–238, Sept 2012.

[Di 2012b] Sheng Di, Derrick Kondo and Walfredo Cirne. *Characterization and comparison of cloud versus Grid workloads.* In International Conference on Cluster Computing (IEEE CLUSTER), pages 230–238, Beijing, China, September 2012.

[Dimitriou 2015] Ioannis Dimitriou, Sara Alouf and Alain Jean-Marie. *A Markovian Queueing System for Modeling a Smart Green Base Station.* In Proceedings of EPEW: European Performance Evaluation Workshop, volume 9272 of *LNCS*, pages 3–18, Madrid, Spain, August 2015.

[Downey 1999] Allen B. Downey and Dror G. Feitelson. *The elusive goal of workload characterization.* j-SIGMETRICS, vol. 26, no. 4, pages 14–29, March 1999.

[Dudin 2013] A. Dudin and V. Klimenok. *Retrial queue of BMAP/PH/N type with customers balking, impatience and non-persistence.* In 2013 Conference on Future Internet Communications (CFIC), pages 1–6, May 2013.

[Gandhi 2010] Anshul Gandhi, Mor Harchol-Balter and Ivo Adan. *Server Farms with Setup Costs.* Perform. Eval., vol. 67, no. 11, pages 1123–1138, November 2010.

[Ghiassi-Farrokhfal 2015] Yashar Ghiassi-Farrokhfal, Srinivasan Keshav, Catherine Rosenberg and Florin Ciucu. *Solar Power Shaping: An Analytical Approach.* IEEE Transactions on Sustainable Energy, vol. 6, no. 1, pages 162–170, January 2015.

[Gianfreda 2014] Marco Gianfreda, Marco Miozzo and Michele Rossi. *SolarStat: Modeling Photovoltaic Sources through Stochastic Markov Processes.* Online, 2014. http://www.dei.unipd.it/∼rossi/Software/Sensors/SolarStat.zip.

[Goiri 2012] Íñigo Goiri, Kien Le, Thu D. Nguyen, Jordi Guitart, Jordi Torres and Ricardo Bianchini. *GreenHadoop: Leveraging Green Energy in Data-processing Frameworks.* In Proceedings of the 7th ACM European Conference on Computer Systems, EuroSys '12, pages 57–70, New York, NY, USA, 2012. ACM.

[Goiri 2015] Íñigo Goiri, Md E. Haque, Kien Le, Ryan Beauchea, Thu D. Nguyen, Jordi Guitart, Jordi Torres and Ricardo Bianchini. *Matching Renewable Energy Supply and Demand in Green Datacenters.* Ad Hoc Netw., vol. 25, no. PB, pages 520–534, February 2015.

[Greenberg 2008] Albert Greenberg, James Hamilton, David A. Maltz and Parveen Patel. *The Cost of a Cloud: Research Problems in Data Center Networks.* SIGCOMM Comput. Commun. Rev., vol. 39, no. 1, pages 68–73, December 2008.

[Gu 2001] Lianhong Gu, Jose D. Fuentes, Michael Garstang, Julio Tota da Silva, Ryan Heitz, Jeff Sigler and Herman H. Shugart. *Cloud modulation of surface solar irradiance at a pasture site in southern Brazil.* Agricultural and Forest Meteorology, vol. 106, no. 2, pages 117–129, January 2001.

[Guo 2014] Lizheng Guo, Tao Yan, Shuguang Zhao and Changyuan Jiang. *Dynamic Performance Optimization for Cloud Computing Using M/M/m Queueing System.* Journal of Applied Mathematics, vol. vol. 2014, page 8 pages, 2014.

[Harchol-Balter 2013] Mor Harchol-Balter. Performance modeling and design of computer systems: Queueing theory in action. Cambridge University Press, New York, NY, USA, 1st édition, 2013.

[He 2000] Qi-Ming He, Hui Li and Yiqiang Q. Zhao. *Ergodicity of the BMAP/PH/s/s+K retrial queue with PH-retrial times.* Queueing Systems, vol. 35, no. 1, pages 323–347, Jul 2000.

[Horváth 2002] András Horváth and Miklós Telek. *PhFit: A General Phase-Type Fitting Tool.* In Proceedings of TOOLS: International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, volume 2324 of *LNCS*, pages 82–91, London, UK, April 2002.

[Horváth 2013a] Gábor Horváth and Hiroyuki Okamura. *A Fast EM Algorithm for Fitting Marked Markovian Arrival Processes with a New Special Structure.* In Maria Simonetta Balsamo, William J. Knottenbelt and Andrea Marin, editors, Computer Performance Engineering, pages 119–133, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Horváth 2013b] Gábor Horváth and Hiroyuki Okamura. *A Fast EM Algorithm for Fitting Marked Markovian Arrival Processes with a New Special Structure.* In Computer Performance Engineering, pages 119–133. Springer Berlin Heidelberg, 2013.

[Hu 2008] Liting Hu, Hai Jin, Xiaofei Liao, Xianjie Xiong and Haikun Liu. *Magnet: A novel scheduling policy for power reduction in cluster with virtual machines.* In 2008 IEEE International Conference on Cluster Computing, pages 13–22, Sept 2008.

[Hulstrom 2003] Roland L. Hulstrom, editor. Solar resources. MIT Press, 2003.

[Iqbal 1983] Muhammad Iqbal. An introduction to solar radiation. Academic Press, 1983.

[Jackson 2001] David B. Jackson, Quinn Snell and Mark J. Clement. *Core Algorithms of the Maui Scheduler.* In Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing, JSSPP '01, pages 87–102, London, UK, UK, 2001. Springer-Verlag.

[Jain 1991] Raj Jain. The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling. Wiley professional computing. Wiley, 1991.

[Jean-Marie 2017] Alain Jean-Marie. *marmoteCore: a Markov Modeling Platform.* In VALUETOOLS: Performance Evaluation Methodologies and Tools, Venice, Italy, December 2017.

[Jurado 1995] M. Jurado, J.M. Caridad and V. Ruiz. *Statistical distribution of the clearness index with radiation data integrated over five minute intervals.* Solar Energy, vol. 55, no. 6, pages 469–473, December 1995.

[Kanungo 2002] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman and Angela Y. Wu. *An efficient k-means clustering algorithm: analysis and implementation.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 7, pages 881–892, July 2002.

[Kim 2008] Chesoong Kim, Valentina I. Klimenok and Dmitry S. Orlovsky. *The BMAP/PH/N retrial queue with Markovian flow of breakdowns.* European Journal of Operational Research, vol. 189, no. 3, pages 1057 – 1072, 2008.

[Kim 2010] Che Soong Kim, Valentina Klimenok, Vilena Mushko and Alexander Dudin. *The BMAP/PH/N retrial queueing system operating in Markovian random environment.* Computers & Operations Research, vol. 37, no. 7, pages 1228 – 1237, 2010. Algorithmic and Computational Methods in Retrial Queues.

[Kim 2013] Che Soong Kim, Valentina Klimenok and Alexander Dudin. *Retrial Queueing System with Correlated Input, Finite Buffer, and Impatient Customers.* In Alexander Dudin and Koen De Turck, editors, Analytical and Stochastic Modeling Techniques and Applications, pages 262–276, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Kim 2017] Chesoong Kim, V.I. Klimenok and A.N. Dudin. *Analysis of unreliable BMAP/PH/N type queue with Markovian flow of breakdowns.* Applied Mathematics and Computation, vol. 314, pages 154 – 172, 2017.

[Klimenok 2018] V.I. Klimenok, A.N. Dudin and K.E. Samouylov. *Analysis of the BMAP/PH/N queueing system with backup servers.* Applied Mathematical Modelling, vol. 57, pages 64 – 84, 2018.

[Krishnamoorthy 2008] A. Krishnamoorthy, S. Babu and Viswanath C. Narayanan. *MAP/(PH/PH)/c Queue with Self-Generation of Priorities and Non-Preemptive Service.* Stochastic Analysis and Applications, vol. 26, no. 6, pages 1250–1266, 2008.

[Liu 2012a] Z. Liu and S. Cho. *Characterizing Machines and Workloads on a Google Cluster.* In 2012 41st International Conference on Parallel Processing Workshops, pages 397–403, Sept 2012.

[Liu 2012b] Zhenhua Liu, Yuan Chen, Cullen Bash, Adam Wierman, Daniel Gmach, Zhikui Wang, Manish Marwah and Chris Hyser. *Renewable and Cooling Aware Workload Management for Sustainable Data Centers.* In Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '12, pages 175–186, New York, NY, USA, 2012. ACM.

[Ltd.] Solarbotics Ltd. *SCC-3733 Monocrystalline solar cells.*

[Lucantoni 1991] David M. Lucantoni. *New results on the single server queue with a batch markovian arrival process.* Communications in Statistics. Stochastic Models, vol. 7, no. 1, pages 1–46, 1991.

[Mary 2012] Nnemeka Ani Brown Mary and Krish Saravanan. *Performance Factors of Cloud Computing Data Centers Using [(M/G/1) : (∞/Gdmodel)] Queuing Systems.* International Journal of Grid Computing & Applications, vol. 4, 10 2012.

[Miozzo 2014] Marco Miozzo, Davide Zordan, Paolo Dini and Michele Rossi. *SolarStat: Modeling Photovoltaic Sources through Stochastic Markov Processes.* In Proc. of 2014 IEEE International Energy Conference, pages 688–695, Dubrovnik, Croatia, May 2014.

[Mishra 2010] Asit K. Mishra, Joseph L. Hellerstein, Walfredo Cirne and Chita R. Das. *Towards Characterizing Cloud Backend Workloads: Insights from Google Compute Clusters.* SIGMETRICS Perform. Eval. Rev., vol. 37, no. 4, pages 34–41, March 2010.

[Mitrani 2011] Isi Mitrani. *Service center trade-offs between customer impatience and power consumption.* Performance Evaluation, vol. 68, no. 11, pages 1222 – 1231, 2011. Special Issue: Performance 2011.

[Mitrani 2013a] Isi Mitrani. *Managing performance and power consumption in a server farm.* Annals of Operations Research, vol. 202, no. 1, pages 121–134, Jan 2013.

[Mitrani 2013b] Isi Mitrani. *Trading Power Consumption against Performance by Reserving Blocks of Servers.* In Mirco Tribastone and Stephen Gilmore, editors, Computer Performance Engineering, pages 1–15, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Neglia 2016] Giovanni Neglia, Matteo Sereno and Giuseppe Bianchi. *Geographical Load Balancing across Green Datacenters.* ACM SIGMETRICS Performance Evaluation Review, vol. 44, no. 2, pages 64–69, September 2016.

[Okamura 2009] Hiroyuki Okamura and Tadashi Dohi. *Faster maximum likelihood estimation algorithms for Markovian arrival processes.* In Quantitative Evaluation of Systems, QEST'09, pages 73–82, 2009.

[Oró 2015] Eduard Oró, Victor Depoorter, Albert Garcia and Jaume Salom. *Energy efficiency and renewable energy integration in data centres. Strategies and modelling review.* Renewable and Sustainable Energy Reviews, vol. 42, no. C, pages 429–445, 2015.

[Panasonic] Panasonic. *Panasonic HIT.*

[Piedallu 2007] Christian Piedallu and Jean-Claude Gégout. *Multiscale computation of solar radiation for predictive vegetation modelling.* Annals of Forest Science, vol. 64, no. 8, pages 899–909, January 2007.

[Politaki 2017] Dimitra Politaki and Sara Alouf. *Stochastic Models for Solar Power.* In Philipp Reinecke and Antinisca Di Marco, editors, Computer Performance Engineering, pages 282–297, Cham, 2017. Springer International Publishing.

[ptaff.ca] ptaff.ca. *Sunrise, sunset daylight in a graph.*

[Reiss 2011] Charles Reiss, John Wilkes and Joseph L. Hellerstein. *Google cluster-usage traces: format + schema.* Technical report, Google Inc., Mountain View, CA, USA, November 2011. Revised 2014-11-17 for version 2.1. Posted at https://github.com/google/cluster-data.

[Reiss 2012] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz and Michael A. Kozuch. *Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis.* In Proceedings of the Third ACM Symposium on Cloud Computing, SoCC '12, pages 7:1–7:13, New York, NY, USA, 2012. ACM.

[Sakuma 2010] Yutaka Sakuma. *Asymptotic behavior for MAP/PH/c queue with shortest queue discipline and jockeying.* Operations Research Letters, vol. 38, no. 1, pages 7 – 10, 2010.

[Schwartz 2012] C. Schwartz, R. Pries and P. Tran-Gia. *A queuing analysis of an energy-saving mechanism in data centers.* In The International Conference on Information Network 2012, pages 70–75, Feb 2012.

[Sharma 2011] Bikash Sharma, Victor Chudnovsky, Joseph L. Hellerstein, Rasekh Rifaat and Chita R. Das. *Modeling and Synthesizing Task Placement Constraints in Google Compute Clusters.* In Proceedings of the 2Nd ACM Symposium on Cloud Computing, SOCC '11, pages 3:1–3:14, New York, NY, USA, 2011. ACM.

[Staples 2006] Garrick Staples. *TORQUE Resource Manager.* In Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, SC '06, New York, NY, USA, 2006. ACM.

[S.V. Garimella 2013] J. Weibel L.-T. Yeh S.V. Garimella T. Persoons. *Technological Drivers in Data Centers and Telecom Systems: Multiscale Thermal, Electrical, and Energy Management.* Applied Energy, vol. 107, pages 66–88, 2013.

[Van Heddeghem 2014] 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, vol. 50, pages 64–76, September 2014.

[Wang 2011] Kai Wang, Steven Low and Chuang Lin. *How stochastic network calculus concepts help green the power grid.* IEEE SmartGridComm, pages 55–60, October 2011.

[Wang 2012] Kai Wang, Florin Ciucu, Chuang Lin and Steven H. Low. *A stochastic power network calculus for integrating renewable energy sources into the power grid.* IEEE Journal on Selected Areas in Communications, vol. 30, no. 6, pages 1037–1048, July 2012.

[Wang 2015] Kai Wang, Minghong Lin, Florin Ciucu, Adam Wierman and Chuang Lin. *Characterizing the Impact of the Workload on the Value of Dynamic Resizing in Data Centers.* Perform. Eval., vol. 85, no. C, pages 1–18, March 2015.

[Wolff 1989] R.W. Wolff. Stochastic modeling and the theory of queues. Prentice-Hall international series in industrial and systems engineering. Prentice Hall, 1989.

[Íñigo Goiri 2015] Íñigo Goiri, Md E. Haque, Kien Le, Ryan Beauchea, Thu D. Nguyen, Jordi Guitart, Jordi Torres and Ricardo Bianchini. *Matching renewable energy supply and demand in green datacenters.* Ad Hoc Networks, vol. 25, pages 520 – 534, 2015. New Research Challenges in Mobile, Opportunistic and Delay-Tolerant Networks Energy-Aware Data Centers: Architecture, Infrastructure, and Communication.