CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Combinatorial optimisation for sustainable cloud computing
Authors	De Cauwer, Milan
Publication date	2018
Original Citation	De Cauwer, M. 2018. Combinatorial optimisation for sustainable cloud computing. PhD Thesis, University College Cork.
Type of publication	Doctoral thesis
Rights	© 2018, Milan De Cauwer. - http://creativecommons.org/licenses/by-nc-nd/3.0/
Download date	2026-11-01 12:45:45
Item downloaded from	https://hdl.handle.net/10468/6903



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Combinatorial Optimisation for Sustainable Cloud Computing

Milan De Cauwer
MSc

Thesis submitted for the degree of
Doctor of Philosophy



NATIONAL UNIVERSITY OF IRELAND, CORK

COLLEGE OF SCIENCE, ENGINEERING AND FOOD SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

August, 2018

Head of Department: Prof Cormac Sreenan

Supervisors: Prof Barry O'Sullivan
Dr Deepak Mehta

Contents

List of Figures . . . iv
List of Tables . . . vi
Abstract . . . vii
Declaration . . . viii
Acknowledgements . . . xi

1 Introduction 1
1.1 Context . . . 1
1.2 Cloud Computing, Data Centres and Workload Management Systems. 3
1.3 Key Challenges . . . 5
1.4 Thesis Statement and Contributions . . . 6
1.5 Structure of the Dissertation . . . 8

2 Background, Related Work and Datasets 10
2.1 Overview . . . 11
2.2 Constraint Optimisation Problems . . . 12
2.2.1 Definitions . . . 12
2.2.2 Optimal and Sub-optimal Solutions . . . 15
2.2.3 Online, Semi-online and Offline Optimisation . . . 17
2.2.4 Modeling and Solving Constraint Optimisation Problems . . . 18
2.2.4.1 Front-end Modeling Tools . . . 18
2.2.4.2 Back-end Solvers . . . 19
2.3 Optimisation Problems in Cloud Computing Systems . . . 21
2.3.1 Cost Models for Data Centres . . . 22
2.3.2 Managing Workloads in Data Centres . . . 23
2.3.3 Workload Consolidation and Virtualisation Technologies . . . 23
2.3.4 Workload Consolidation as a Packing Problem . . . 24
2.3.5 Extracting Evaluation Datasets . . . 26
2.3.5.1 Data Extraction . . . 26
2.3.5.2 Characterising the Workload . . . 28
2.4 Workload Management in Geographically Distributed Clouds . . . 33
2.4.1 Models for Electricity Prices and Price Prediction Errors . . . 35
2.4.2 Minimizing Data Centre Electricity Cost . . . 39
2.4.3 Analysis . . . 41
2.5 Conclusion . . . 47

3 A Generalisation of Bin Packing as a Core Consolidation Problem 48
3.1 The Temporal bin packing Problem . . . 49
3.2 Packing Versus Temporal Models . . . 52
3.2.1 Packing Model (PA) . . . 52
3.2.2 Temporal Model (TP) . . . 54
3.3 Breaking Symmetry . . . 56
3.3.1 Breaking Symmetry on the PA model. . . . 57
3.3.2 Breaking Symmetry on the TP model. . . . 57
3.4 Lower and Upper Bounds . . . 58

3.5 Empirical Analysis . . . 60
3.5.1 Experimental setup . . . 61
3.5.2 Instances . . . 61
3.5.3 Analysis . . . 62
3.6 Conclusion and Limitations . . . 64

4 Semi-online Consolidation with Uncertain Task Duration 67
4.1 Semi-Online Resource Wastage Minimisation . . . 68
4.1.1 The Semi-Online Framework . . . 71
4.1.2 The Monitor Module . . . 71
4.1.3 The Solver Module . . . 72
4.1.4 Illustrating the Consolidation of Machine Run Times . . . 74
4.2 Packing Heuristics . . . 75
4.3 A novel placement policy: First Merged Fit (FMF) . . . 77
4.3.1 Illustration . . . 77
4.3.2 The First Merged Fit Algorithm . . . 77
4.4 Local Search . . . 80
4.5 Empirical Analysis . . . 80
4.5.1 Overall Allocated Resources . . . 81
4.5.2 Resource Allocation During Peak Activity Periods . . . 82
4.5.3 Resource Allocation under Varying Time Step Duration . . . 85
4.5.4 Resource Allocation under Uncertain Task Duration . . . 87
4.5.5 Real-time Placement of Incoming Tasks . . . 89
4.5.6 Comparing Policies . . . 91
4.6 Conclusion and Limitations . . . 93

5 Online Consolidation with Uncertain Task Sizes 95
5.1 Methodology . . . 97
5.2 Prediction Module . . . 98
5.2.1 Input . . . 100
5.2.2 Output . . . 101
5.3 Scheduling Module . . . 103
5.3.1 Mathematical Model . . . 104
5.3.2 Policies for Online Scheduling . . . 106
5.4 Monitoring Module . . . 109
5.5 Experiments . . . 112
5.5.1 Experimental Setup and Error Metrics . . . 113
5.5.2 Predicting CPU and RAM Maximum Utilisation . . . 114
5.5.3 Evaluating the Scheduling Policies . . . 115
5.5.3.1 Polices for Known Peak Resource Requirements . . . 116
5.5.3.2 Policy-Predictor Interactions . . . 118
5.5.3.3 Eviction policies . . . 122
5.6 Conclusion and Limitations . . . 123

6 Proactive Consolidation with VM Migrations 124
6.1 The Proactive Workload Consolidation Problem . . . 125
6.2 An Integer Linear Model for the PWCP . . . 127

6.3 Empirical Analysis 129
6.4 Conclusion and Limitations 132

7 On Bin Packing Instances **134**
7.1 The Weibull Distribution 135
7.2 Fitting Weibull Distributions to Real-world Instances 137
7.2.1 An Example Problem in Data Centre Management 137
7.2.2 Verifying the Goodness-of-Fit 139
7.3 Systematic Search for Bin Packing 140
7.3.1 Bin Packing Instances and Solver 140
7.3.2 Small Weibull Shape Parameter Values 142
7.3.3 Full Range of Shape Parameters 144
7.4 Bin Packing Heuristics 145
7.5 Conclusion and Limitations 147

8 Conclusions and Further Work **150**
8.1 Conclusion 150
8.2 Future Work 151
8.2.1 Workload Consolidation as a Component to Complex Cloud Systems 151
8.2.2 Competitive Analysis 152
8.2.3 Understanding Prediction/Optimisation Interactions 152

List of Figures

2.1 Visual illustrating the bin packing problem 15
2.2 Visual illustrating a heuristic solution against the optimal solution . . 16
2.3 Visual illustrating a semi-online solution structure for bin packing . . 18
2.4 A Minizinc model for bin packing 20
2.5 A Numberjack model for bin packing 21
2.6 Histogram analysis of non-utilised resources. 27
2.7 Relationship between requested resources and the duration of the tasks 28
2.8 Characterising jobs arrival rate 30
2.9 Tasks count per job 31
2.10 Number of incoming tasks over elapsed time 31
2.11 Distributions of incoming tasks in the dataset 32
2.12 Showing steps of the Box-Jenkins method to model electricity prices. 37
2.13 Visual comparing actual price data against various models. 38
2.14 An example of the workload dispatched over several data centres. . . 43
2.15 Exploiting price differentials to reduce overall operating costs. 44
2.16 Average optimal assignment cost under several time lags configurations 45
2.17 Average optimal assignment cost under various reconfiguration times. 47

3.1 Comparing solution structures of BP against TBP 51
3.2 A relaxed version of TBP with breakable items 59
3.3 Average gap to lb after 2 and 300 seconds on random instances 63
3.4 Average gap to lb after 2 and 300 seconds on Google instances 65

4.1 Optimal placement considering respectively an on-line, a semi-online (time window of 3s), and a off-line contexts. 70
4.3 The semi-online framework 71
4.4 Two valid assignments of tasks $\{a_1, \ldots, a_5\}$ to 3 standby machines $\{m_0, \ldots, m_2\}$. All tasks are starting at the current time step t. 74
4.5 A run of First Merged Fit (FMF). 78
4.6 Total allocated resources over the time per policies against elapsed time. 82
4.7 Number of allocated machines over the peaks of the 123^{rd} hour. . . . 83
4.8 Allocated resources per policies when increasing time step duration . 86
4.9 Allocated resources while increasing tasks duration uncertainty 88
4.10 Solving time (in seconds) of the placement policies when number of incoming tasks > 100. 90
4.11 Performance of the policies while varying different parameters 92

5.1 Illustration of the experimental Setup 99
5.2 MAE and RMSE values for all the CPU predictors and user defined limit 115
5.3 MAE and RMSE values for all the RAM predictors and user defined limit 116
5.4 Comparing the Random and Round Robin polices on the with and without using the greedy scheme. 117
5.5 Illustration of the number of active machines in time. 118
5.6 Illustration of of the average CPU and RAM utilisation. 119

5.7 Aggregated CPU peaks. 121
5.8 Performance of the eviction policies across different classes of priority. 122

6.1 Visual illustrating a solution of PWCP 127
6.2 Varying window size w over selected values of N. Each value of $w \in \{0, 2, 4, 6, 8, 10\}$ corresponds to a different line style. 129
6.3 Varying the migration limits (k) over selected values of w. Each value of $k \in \{0, 2, 4, 6, 8, 10\}$ corresponds to a different line style. 130
6.4 Visual on the trade-off between energy cost and QoS 131
6.5 VC function of N and w 132

7.1 Weibull distributions 136
7.2 An example of the quality of fit using a Weibull distribution 138
7.3 Average runtime and percentage of solved instances. 143
7.4 Average runtime and percentage of solved instances exhibiting the easy-hard-easy behaviour in search effort. 146
7.5 Average number of bins in optimal solutions 147
7.6 The difference in the average number of bins required by each of the heuristics and the optimal solutions 148

List of Tables

2.1 Seasonal ARIMA model for the electricity price. 39
2.2 Notations for parameters and decision variables 40
2.3 An example of data centre setups. 41

3.1 Percentage of instances solved within the time out function of the symmetry breaking technique 62

4.1 Total run-time of allocated machines (in hours) of the placement policies 81
4.2 Run-time of allocated machines above 180 machines during the 123^{rd} hour. 84
4.3 Resource utilisation (in hours) for time step duration 0, 2 and 30 seconds 87
4.4 Total run-time of allocated machines (in hours) of the placement policies tw=2 89

5.1 Machine usage statistic for Random, Round Robin policies and their restricted counterparts under perfect information from the predictors. . 117
5.2 Waiting time and number of evictions for tasks of different priorities. . 120

7.1 Best-fit Weibull distributions on various instances. 141

Abstract

Enabled by both software and hardware advances, cloud computing has emerged as an efficient way to leverage economies of scale for building large computational infrastructures over a global network. While the cost of computation has dropped significantly for end users, the infrastructure supporting cloud computing systems has considerable economic and ecological costs. A key challenge for sustainable cloud computing systems in the near future is to maintain control over these costs.

Amid the complexity of cloud computing systems, a cost analysis reveals a complex relationship between the infrastructure supporting actual computation on a physical level and how these physical assets are utilised. The central question tackled in this dissertation is how to best utilise these assets through efficient workload management policies. In recent years, workload consolidation has emerged as an effective approach to increase the efficiency of cloud systems. We propose to address aspects of this challenge by leveraging techniques from the realm of mathematical modeling and combinatorial optimisation.

We introduce a novel combinatorial optimisation problem suitable for modeling core consolidation problems arising in workload management in data centres. This problem extends on the well-known bin packing problem. We develop competing models and optimisation techniques to solve this offline packing problem with state-of-the-art solvers. We then cast this newly defined combinatorial optimisation problem in an semi-online setting for which we propose an efficient assignment policy that is able to produce solutions for the semi-online problem in a competitive computational time. Stochastic aspects, which are often faced by cloud providers, are introduced in a richer model. We then show how predictive methods can help decision makers dealing with uncertainty in such dynamic and heterogeneous systems. We explore a similar but relaxed problem falling within the scope of proactive consolidation. This is a relaxed consolidation problem in which one decides which, when and where workload should be migrated to retain minimum energy cost. Finally, we discuss ongoing efforts to model and characterise the combinatorial hardness of bin packing instances, which in turn will be useful to study the various packing problems found in cloud computing environments.

Declaration

This dissertation is submitted to University College Cork, in accordance with the requirements for the degree of Doctor of Philosophy in the Faculty of Science. The research and thesis presented in this dissertation are entirely my own work and have not been submitted to any other university or higher education institution, or for any other academic award in this university. Where use has been made of other people's work, it has been fully acknowledged and referenced. Parts of this work have appeared in the following publications which have been subject to peer review.

This dissertation is borne by the following publications.

1. Ignacio Castiñeiras, Milan De Cauwer, and Barry O'Sullivan. Weibull-based benchmarks for bin packing. In *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, pages 207–222, 2012

2. Milan De Cauwer and Barry O'Sullivan. A study of electricity price features on distributed internet data centers. In Jörn Altmann, Kurt Vanmechelen, and Omer F. Rana, editors, *Economics of Grids, Clouds, Systems, and Services - 10th International Conference, GECON 2013, Zaragoza, Spain, September 18-20, 2013. Proceedings*, volume 8193 of *Lecture Notes in Computer Science*, pages 60–73. Springer, 2013

3. Milan De Cauwer, Deepak Mehta, Barry O'Sullivan, Helmut Simonis, and Hadrien Cambazard. Proactive workload consolidation for reducing energy cost over a given time horizon. In *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2014, Chicago, IL, USA, May 26-29, 2014*, pages 558–561, 2014

4. Jesus Omana Iglesias, Liam Murphy, Milan De Cauwer, Deepak Mehta, and Barry O'Sullivan. A methodology for online consolidation of tasks through more accurate resource estimations. In *Proceedings of the 7th IEEE/ACM International Conference on Utility and Cloud Computing, UCC 2014, London, United Kingdom, December 8-11, 2014*, pages 89–98, 2014

5. Jesus Omana Iglesias, Milan De Cauwer, Deepak Mehta, Barry O'Sullivan, and Liam Murphy. Increasing task consolidation efficiency by using more accurate resource estimations. *Future Generation Comp. Syst.*, 56:407–420, 2016

6. Milan De Cauwer, Deepak Mehta, and Barry O'Sullivan. The temporal bin packing problem: An application to workload management in data centres. In *28th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2016, San Jose, CA, USA, November 6-8, 2016*, pages 157–164. IEEE Computer Society, 2016
7. Vincent Armant, Milan De Cauwer, Kenneth N. Brown, and Barry O'Sullivan. Semi-online task assignment policies for workload consolidation in cloud computing systems. *Future Generation Computer Systems*, 82:89–103, 2018

The contents of this dissertation extensively elaborate upon previously published work and mistakes (if any) are corrected. Some sections of the dissertation are unpublished but may appear in future peer reviewed publications.

Optimisation models, code snippets and data available from:

- `https://gitlab.insight-centre.org/mdecauwer/`

Milan De Cauwer

August, 2018

To all the people met on the way.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my academic supervisors, Prof. Barry O'Sullivan and Dr. Deepak Mehta for the continuous and unconditional support during my studies. For their guidance, professionalism and patience. I am deeply thankful for the opportunity that they provided me with and even more so for their support. My work would not have been possible without the support of Science Foundation Ireland Grant No. 10/IN.1/I3032 and 12/RC/2289 which is co-funded under the European Regional Development Fund.

I would like to acknowledge and thank my coauthors, talented researchers, with whom I had the opportunity to collaborate. Barry and Deepak in the supervising roles but also Vincent Armant, Ken Brown, Helmut Simonis, Jesus Iglesias, Liam Murphy and Nacho Castinieras. I had great pleasure in collaborating with you.

I am grateful to my current and former colleagues here at Insight, all of whom contribute to making Insight a great place to work. My time as a PhD student here at Insight has proved to be a fulfilling life experience not only on a professional but also on a personal level. In no particular order, I wish to personally thank colleagues and friends Barry Hurley, Padraig O'Duinn, Tadhg Fitzgerald, Cathal Hoare and Diarmuid Grimes for being welcoming and making me feel at home here in Ireland. Gilles and Marius for making me a better musician as well as a keen rock-climber. Anne-Marie George, Yves Sohege, Mesut Kaya and Mo Siala for the excitement of being around them.

I sincerely thank staff members, Linda, Caitriona, Eleanor, Peter and Chrys for making sure that every member of the team stays on track.

For their long term and continuing support (30+ years), I am very much indebted to my family without whom I would not be here typing these words. To my mother Beatrice and my father Ronan, to my sisters Isabelle and Aurore, thank you.

Finally, I would like to thank Patrick Healy and Steve Prestwich, respectively external and internal members of the jury, for having taken time to read and comment my dissertation. The corrections that you suggested are certainly making for a better dissertation.

Chapter 1

Introduction

Summary. *This chapter introduces the research axes discussed throughout this dissertation. We first discuss the notion of resource management in cloud computing structures in the emergent context of sustainable computing. We then highlight that powering cloud infrastructures yields challenges that should be resolved. Some key challenges brought by the nature of cloud systems are then defined as a motivation for this dissertation. Finally, we provide a detailed outline of the dissertation.*

1.1 Context

Computational sustainability has emerged as an interesting concept in the field of computer science. It is closely related and inspired by ideas arising in the field of sustainable development and is relevant to many goals set by governments and societies to achieve development today without compromising tomorrow's possibilities [BKA+87]. Computational sustainability can be defined as follows: [1]

"*Computational sustainability is an interdisciplinary field that aims to apply techniques from computer science, information science, operations research, applied mathematics, and statistics for balancing environmental, economic, and societal needs for sustainable development.*"

The era of data-driven decision-making often relies on data centres as a backbone to store, structure and retrieve information for use in cloud computing contexts. Because

[1] http://www.computational-sustainability.org/

these ubiquitous facilities have a large economic and ecological footprint, designing methods to exploit them efficiently is a central challenge in maintaining such structures sustainable. Achieving sustainability is a key challenge for the emergence of data-driven societies.

Organisations implementing cloud computing solutions are emerging as a very compelling answer to the challenge of outsourcing computational tasks and data management needs [BBA10, ZCB10, LCW11]. While the popularity of cloud solutions can be explained by various factors, its development has given rise to many challenges in terms of workload management and scalability of such systems. This dissertation contributes to the effort of formally modeling parts of such systems for optimisation purposes. More specifically, the focus is put on modeling and optimising combinatorial problems using methods and techniques from the realm of mathematical modeling and operations research.

Operations research finds its origin in the efforts put in managing scarce resources during the second World War [Lar84]. The aim of operations research is to assist decision-making in the context of complex systems by leveraging the formalism of mathematical modeling and optimisation techniques. The rather broad range of problems addressed by operations research encompasses project planning, network optimisation, assignment problems and scheduling problems, to name but a few [WG04]. Many techniques have been developed to tackle these problems such as mathematical optimisation over discrete or continuous domains, multi-criteria decision-making or elements of reasoning in stochastic environments. The success of operations research can be seen in many applications arising in real-world environments such as scheduling train timetables [CJT16, CT12], assigning crews to flights [JB09, BJM09] or flow optimisation in networks [LL99].

In real-world environments, decision-makers often face uncertainty in the data defining their problem [Sah04]. While this uncertainty can be understood in terms of the partially missing or partially incorrect information, a significant branch of research is interested in retrieving missing or incorrect information through analysis of historical data. Statistical modeling and machine learning has proved very successful in building strong predictive models from historical data. Its success is due to a relatively low cost of harvesting data and the increasing computational power available to data scientists.

Cloud computing, as a concept and its implementation using data centres gave rise to complex technology-dependent world wide systems. Nonetheless, there is an opportunity for operations research and machine learning to significantly reduce its economic and ecological impact [BBA10]. This is particularly the case on an abstract level deal-

ing with workload management systems in which decisions can be computationally hard. This is therefore the focus of this dissertation.

1.2 Cloud Computing, Data Centres and Workload Management Systems.

Cloud computing can be thought of as a generic term for the delivery of computing power over the Internet [LCW11]. By abstracting the means of computation from what has to be computed, it allows end users to externalise raw computation, storage or routinely running applications to a third-party facility. These services are usually made available to users as a flexible, on-demand utility that can be transparently scaled to meet their computational needs. The National Institute of Standards and Technology defines cloud computing as follows [ZCB10]:

"Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

Although the terminology surrounding cloud computing started to develop in the early 2000's, its underlying concepts can be traced back to the late 1960's and early 1970's. The idea of giving access to a computing facility over a network stems from the birth of ARPANET in 1969 which can be informally thought as the first step toward a global network allowing to access remote resources. Remarkably, the notion of computing over a network as a utility has been discussed as early as 1966 [Par66]. The development of UNIX in the early 1970's started with the goal to allow concurrent users to simultaneously access the resources of a single computer. Subsequently, UNIX systems inspired a vast diversity of multi-tasking, multi-users operating systems such as the family of BSD operating systems and the ubiquitous Linux kernel.

As suggested previously, while the emergence of cloud computing can be attributed to the convergence of several ripening technologies, its success can be understood in terms of compelling features offered to businesses and end users. Among these, the following stand out as very compelling for end users to adopt cloud services more widely. These aspects are discussed in [ZCB10, GHMP09].

Minimal Investment

Using cloud systems as a back-end to offload computational tasks provides the

advantage of requiring a minimal investment underlying IT infrastructure to be used as a front-end to access the cloud. Expensive critical components such as mail servers, web servers or high-throughput computing hardware can be accessed as a utility through the cloud.

Scalability

As the need for computing power may vary over time for end user, scalability and flexibility are a considerable advantage of using cloud based solutions. Cloud users can easily scale their use of computation in a very reactive manner by orders of magnitude.

Minimal Maintenance Cost

The high maintenance cost of the computational facilities is offloaded to a third-party cloud system. Maintenance of both the computing hardware itself and all the equipment needed (redundant power supply, cooling systems, networking systems) is offloaded to the cloud provider.

Cloud computing has been a successful model for computation by abstracting the computational means from the actual computation. On the user side, there is no need to know where the computation will be held, or what kind of hardware will carry the computation. Cloud providers can offer such services by leveraging sizable economies of scale while designing large computing facilities accessible through the cloud. On the provider's side, according to [ZCB10, GHMP09], notable factors for the success of providing computing facilities as a utility are:

Economies of scale

The overall cost of a unit of computation depends on multiple factors. Naturally, fractions of this cost are subject to rather significant economies of scale. This is particularly the case for networking and cooling facilities for which the marginal cost of an additional server is negligible.

Resiliency and Redundancy

Given their critical size, cloud providers are implementing a certain degree of resilience and redundancy across their computing facility. This allows cloud providers to offer a high level of availability of their computing infrastructure.

The computation power offered by cloud providers is usually implemented in data centres hosting both servers used to carry computation along with the equipment needed for their operation. These facilities usually implement a certain degree of redundancy in both power supply and network communication capabilities. Concentrating considerable computational power in one place allows one to fully leverage economies of

scale to reduce the initial investment, operating costs and maintenance cost. Since data centres are the physical backbone hosting the computational power of cloud systems, adequate management of the workload they have to face is critical.

In this dissertation we have a particular focus on workload consolidation problems, sometimes also called resource or server consolidation. It is a technique often leveraged in data centres to increase the overall efficiency of a cluster of physical machines. Workload consolidation techniques may be used to reduce the overall number of servers either owned or required by the cloud provider to meet the demand from end users. The effort in developing consolidation mechanisms is justified by the rather low rate at which servers resources are utilised. In other words, the aim of consolidation is to do *more [work] with less [space/energy]*. Although the concepts behind workload consolidation are rather simple, the implementation is usually constrained by both the technologies at play in the data centre and the particular use-case that the cloud provider is focused on.

1.3 Key Challenges

Formally modeling cloud computing systems exposes a few key challenges that need to be taken into consideration. Due to the heterogeneity of the various use-cases tackled in industry, settling on a unique problem to model and solve is a rather difficult task. The industrial applications of cloud computing cover a large spectrum of use-cases ranging from hosting long running cloud services [PCG+10, BFF+10] to on-demand performance computing [LTC14a, MHL+13]. In addition, cloud computing infrastructures often implement very different hardware technologies. These technologies and standards are changing tremendously on both a software and hardware level. This leads to a large variety of operational settings for which optimisation models can be widely divergent. Appropriately capturing the problems faced by cloud providers can therefore be challenging.

A large variety of problems faced by cloud providers are modeled for optimisation purposes. The nature of these problems are ranging from discrete optimisation for managing workload in geographically distributed data centres [WGM+17, CO13] to continuous non-linear optimisation applied to managing temperatures within a data centres [CCMO15, CSG11]. In addition, it is often the case that goals are conflicting. For instance, many operational setting require techniques from the realm of multi-objective optimisation. The trade-off between how many servers are required to ac-

commodate a given workload against the makespan of the schedule illustrates the need for multi-objective approaches [XF10]. Another example of conflicting objectives is the downtime of servers against the overall maintenance cost of a Cloud system. Finally, the nature of workloads faced by cloud systems is quite often characterised by uncertainties linked to the online nature of consolidation problems on one side and the stochastic nature of some aspects of the tasks to be consolidated. Understanding the stochastic nature of workloads faced by cloud providers is key to drive optimisation processes to achieve better decision-making.

Beyond the large variety of aspects that can be studied for optimisation purposes, combinatorial problems found in cloud systems share the property of having a rather large size [RTG+12]. Due to the effort required by cloud providers to reach a critical operational size, combinatorial aspects related to workload assignment, server placement and cooling are by nature characterised by large instance sizes. Despite the large size of the problems, a desirable aspect is to implement any-time properties in algorithms and procedures. It is indeed crucial for cloud providers to be able to take decisions in a timely manner to be able to cope with high utilisation of their infrastructure [AFG+09].

Within the broad range of optimisation problems discussed in the literature, workload consolidation has emerged as a suitable technique for managing workloads in the context of cloud computing [HI15]. This dissertation therefore focuses on a variety of consolidation problems with the aim of reducing the resource wastage in data centres.

1.4 Thesis Statement and Contributions

The implementation of cloud computing gives rise to complex systems in which many aspects can be optimised [ZCB10]. In an attempt to bring elements of sustainability to cloud computing infrastructure this dissertation will be centered on formalising and solving combinatorial problems with a particular focus on workload consolidation problems. In the following, we state the thesis defended in this dissertation along with the contributions it makes to the field.

Sub-thesis 1. *To date a full body of literature is focused on optimising energy consumption in cloud computing infrastructures. Many approaches do rely on well-known combinatorial problems, either studied on their own or in the context of an other application. We claim that the bin packing problem is a natural view on many workload consolidation problems. However, novel generalisations of the classic bin packing problem are needed to capture the sophistication of real-world data centres setting.*

Contributions. Chapter 3 employs mathematical modeling to formally express a core workload consolidation problem found in many applications pertaining to cloud systems. We explore a number of techniques from the realm of operations research and constraint programming to solve this core consolidation problem in an offline context. These contributions have appeared in the following publication:

> Milan De Cauwer, Deepak Mehta, and Barry O'Sullivan. The temporal bin packing problem: An application to workload management in data centres. In *28th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2016, San Jose, CA, USA, November 6-8, 2016*, pages 157–164. IEEE Computer Society, 2016.

Chapter 4 expands on the latter by casting it in a semi-online setting in which information is only partially known at decision time. We present a set of solving approaches tailored to finding solutions to the consolidation problem under severe computational time limits. Although the latter aspect is often ignored in the literature, cloud providers need to implement fast decision-making policies. These aspects are discussed in the following journal paper:

> Vincent Armant, Milan De Cauwer, Kenneth N. Brown, and Barry O'Sullivan. Semi-online task assignment policies for workload consolidation in cloud computing systems. *Future Generation Computer Systems*, 82:89–103, 2018.

Sub-thesis 2 *A common issue in daily operations in data centres is dealing with incomplete or even missing information regarding the nature of the workload that needs to be consolidated. We claim that some missing information can be at least partially predicted with machine learning techniques and successfully used within optimisation processes.*

Contributions. In Chapter 5 we discuss a consolidation problem for which uncertainties are coming from unknown object sizes at decision time. The specific application studied in this chapter needs to take into account object eviction to avoid over allocating machines. We explore the interactions between standard machine learning models used to retrieve missing information and the outcome of consolidation policies. These contributions appear as a comprehensive article in the following journal:

> Jesus Omana Iglesias, Milan De Cauwer, Deepak Mehta, Barry O'Sullivan, and Liam Murphy. Increasing task consolidation efficiency by using more accurate resource estimations. *Future Generation Comp. Syst.*, 56:407–420, 2016.

In Chapter 6 the nature of the uncertainty differs from previous chapter in the sense that the size of the objects is varying in time. The information available at decision time is somehow restricted by the online nature of the problem. In this chapter we discuss a rather different consolidation problem in which workload can be migrated in order to retain a minimum energy cost while satisfying operational constraints. The latter contributions have been published as a short paper as follows:

> Milan De Cauwer, Deepak Mehta, Barry O'Sullivan, Helmut Simonis, and Hadrien Cambazard. Proactive workload consolidation for reducing energy cost over a given time horizon. In *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2014, Chicago, IL, USA, May 26-29, 2014*, pages 558–561, 2014.

This dissertation aims to defend these claims. The next sections provides a detailed outline of the contents of each chapters.

1.5 Structure of the Dissertation

This dissertation addresses the challenges discussed in the previous section and will be structured as follows. Chapter 2 provides an overview of the area in which our work has been carried. We provide background information on techniques for modeling and solving combinatorial problems. We then discuss organisational aspects of cloud computing systems and review efforts made to optimise various facets of such infrastructures. A particular focus will be put on workload consolidation problems and relevant techniques developed to solve them. The variety of consolidation problems tackled throughout this dissertation is rather broad in the settings they are considering. Therefore each contribution chapter features its own set of notation.

Chapter 3 formalises a packing problem that emerges as a core sub-problem for managing workload consolidation in data centres. We introduce this packing problem as a generalisation of the bin packing problem in which items have a lifespan and the objective function is aiming at minimising some notion of resource wastage. Contrasting optimisation models using Mixed Integer Programming and Constraint Programming were developed and studied.

Chapter 4 expands on the previously defined packing problem by casting the static consolidation problem in a semi-online setting. Because of the online nature of consolidation problems, we are interested in developing online or semi-online policies for workload consolidation allowing one to compute a feasible allocation plan in a short

computational time. We introduce a semi-online formalisation of the core problem discussed in the previous chapter and introduce an assignment policy that exploits properties of this formulation. In addition, we systematically study two aspects pertaining to real-world implementations of workload consolidation problems. We first relax the assumption of perfect knowledge of task duration and study the behavior of policies under imperfect information. The second aspect studied is the trade-off between delay in processing the workload against the quality of consolidation policies.

In Chapter 5, we introduce elements of stochasticity. For the sake of developing more realistic consolidation environments, we explore a model in which the size of tasks is not fully known at decision time. Our methodology uses elements of machine learning in order to accurately predict the size of the tasks to be assigned. So formulated, the problem is closer to traditional scheduling problems rather than packing problems. We propose a methodology for achieving an efficient utilisation of a cluster's resources while providing users with fast and reliable computing services. The methodology consists of three main modules: i) a prediction module that forecasts the maximum resource requirement of a task; ii) a scheduling module that efficiently consolidates the workload; and iii) a monitoring module that tracks the levels of utilisation of the machines and tasks, and can evict one or more tasks from the machines.

In Chapter 6, we explore a relaxed consolidation problem in which we allow migration of tasks to take place. Although migrations come at the cost of introducing latency in the system, it allows for more efficient consolidation techniques. In the considered model, information available at decision time is only partial and limited to an arbitrarily small rolling window of time. We perform investigations to understand the relationship between the number of time-periods considered in one optimisation step and migration-limits on the Service Level Agreements, energy cost, server-transition cost and migration cost. Our results suggest that looking ahead by only a few time-periods can lead to significantly more efficient resource provisioning over the entire horizon and consequently higher energy efficiency and close to no service level violations.

In Chapter 7, we present a parameterisable benchmark generator for bin packing instances based on the well-known Weibull distribution. Using the shape and scale parameters of this distribution we can generate benchmarks that contain a variety of item size distributions. We show that real-world bin packing benchmarks can be modeled extremely well using our approach. We also study both systematic and heuristic bin packing methods under a variety of Weibull settings.

Finally, in Chapter 8, we summarise our work and contributions and provide directions that would be worthy to explore as future work.

Chapter 2

Background, Related Work and Datasets

Summary. *This chapter introduces background information and related work revolving around the notions of combinatorial optimisation and cloud computing technologies. We first introduce constraint optimisation problems and review various techniques to model and solve them. More specifically, we discuss concepts encompassing optimal versus heuristic methods to solve combinatorial problems and some elements of online versus offline optimisation. Several solving paradigms, along with front-end and back-end toolkits used in this dissertation are then presented.*

We then discuss structural aspects of modern cloud computing systems and how their underlying mechanisms are implemented in data centres. We review and discuss efforts made by the research community to model complex cloud systems for optimisation purposes. We put a particular focus on energy consumption models and workload management techniques with a emphasis on workload consolidation strategies.

Concepts from both the realm of optimisation techniques and from cloud computing are then mobilised in a case study showing the importance of appropriately managing workloads in distributed cloud systems. We show how workload can be balanced among geographically distributed data centres in order to take advantage of price differentials on local electricity markets.

2.1 Overview

In this chapter we introduce concepts needed to understand the formalism of problem modeling and problem solving using tools such as mathematical modeling and constraint optimisation. We discuss concepts related to problem modeling, online versus offline optimisation and systematic versus heuristic search. Although these concepts will be used as tools to model and solve workload allocation problems in cloud computing, we aim at providing the necessary background to the reader to understand how these concepts should be leveraged. The concepts presented here allow further discussions to be self-contained. We use the ubiquitous bin packing problem as an illustration of these concepts.

We then provide background information on cloud computing and its structural organisation. We highlight where efforts have been made to reduce the footprint of such systems. The work presented in this dissertation focuses on various workload consolidation problems emerging from modern cloud systems. A large body of literature arrose from studying a variety of workload management problems arising in cloud systems. Amongst the various aspects considered for optimisation are energy-aware, latency-aware, privacy-aware workload management techniques to name but a few. While settling for one particular family of models is rather difficult because of the heterogeneity of technologies and use-cases that can be found in cloud systems, we will focus on rather high-level workload management policies with the aim of minimising their energy footprints.

The last section of this chapter showcases the use of optimisation techniques coupled with statistical learning to minimise the overall electricity bill of a network of interconnected data centres owned by a cloud provider. Many modern cloud systems are provided using distributed data centres that are geographically spread over various locations. Since the energy requirement of these systems is considerable, there is an incentive to ensure that opportunities to access low-cost energy are exploited. The underlying idea in this case-study is to distribute the workload faced by the cloud providers to different locations where energy can be sourced in more affordable markets. Decisions are mostly driven by local electricity market conditions. We explore the impact of the level of price variability, time lag between locations due to the geographical distribution, reconfiguration delay and accuracy of price predictions on the overall electricity cost of managing the workload.

2.2 Constraint Optimisation Problems

Many real-world problems can be successfully tackled using the formalism of mathematical modeling. Applications are ranging from modeling complex biological systems [WW11] to modeling and optimising routing problems [PGGM13, ES10]. In particular, tackling decision and optimisation problems has been the focus of operations research in an effort to describe systems in a rigorous way. In doing so, the model can be studied in order to understand, make predictions, or optimise the underlying system.

Constraint optimisation is a convenient and powerful paradigm to express and optimise real-world combinatorial problems [RBW06]. For instance, the family of planning and scheduling problems are typically modeled using the formalism of constraint optimisation [Tim02]. This formalism will be used in this dissertation to study workload management problems arising in cloud system.

2.2.1 Definitions

Constraint programming is a generic framework used to formally describe and solve combinatorial problems[RBW06]. Using this framework, one aims at finding a complete assignment from values to variables while satisfying a set of constraints. Due to their combinatorial complexity, solving such problems require the use of a search algorithm implementing some backtracking mechanism. Constraint programming can be used to approach both satisfaction and optimisation problems. The remainder of this section provides details on the previous distinction.

Solving Combinatorial Satisfaction Problems (CSPs) requires finding a value for each variable such that all constraints are satisfied. In contrast, to solve Combinatorial Optimisation problems one aims at finding the best solution that optimises a function defined over the problem's variables. Solving a COP requires one to find a complete assignment from values to variables in such a way that the objective function is either minimised or maximised depending on the nature of the problem at hand. Naturally, the assignment is considered valid only if it does satisfy all the constraints stated in the problem.

Definition 1. *Constraint Optimisation Problem. A COP $\mathcal{P}$ is a tuple $\langle \mathcal{X}, \mathcal{D}, \mathcal{C}, f \rangle$ where $\mathcal{X} = \{\mathcal{X}_1, \ldots, \mathcal{X}_n\}$ is a finite set of n variables, $\mathcal{D} = \{\mathcal{D}(\mathcal{X}_1), \ldots, \mathcal{D}(\mathcal{X}_n)\}$ the set of domains where $\mathcal{D}(\mathcal{X}_i)$ is a set of values that can be assigned to variable $\mathcal{X}_i$. The set of k constraints $\mathcal{C} = \{c_1, \ldots, c_k\}$ binds the variables together. Finally, the*

objective function $f \mapsto \mathbb{R}$ is defined over the set of variables.

In a generic form, a combinatorial optimisation problem can be expressed as:

$$\min f(\mathcal{X})$$
$$\text{Subject to}$$
$$\mathcal{X} \in \mathcal{D}(\mathcal{X}_i).$$

The various combinatorial optimisation problems discussed in this dissertation are dealing with finite discrete domains. This is due to the strong combinatorial packing/assignment aspect in variety of workload management problems considered here.

We discuss the bin packing problem to illustrate the aforementioned concepts. The one-dimensional bin packing problem is ubiquitous in operations research. It is defined as follows. Given a set $S = \{s_1, \ldots, s_n\}$ of n indivisible items, each of a known positive size s_i, and m bins of capacity C, the challenge is to decide whether we can pack all n items into the m bins such that the sum of sizes of the items hosted in each bin does not exceed the bin's capacity C. In the canonical formulation, the bins are considered to be unit sized (i.e. $C = 1.0$). The one-dimensional bin packing problem is NP-Complete [CCG+13]. Since it was first discussed, it has given rise to a full body of literature exploring variants and generalisations of the one-dimensional case. For instance, natural extensions to multiple dimensions are discussed in [CK04]. A classification of the variants of packing problems is provided here [Dyc90]. Among the many applications of this problem are timetabling, scheduling, stock cutting, television commercial break scheduling, and container packing [CO10, dC98]. Higher dimensional bin packing problems are highly relevant to real-world application. The two-dimensional variant, also known as rectangle packing has been studied here [SO11, SO08, Vid04, DSD10]. Similarly, the three-dimensional case has many industrial applications.

Typical bin packing methods rely on either heuristics [ARGA04], meta heuristics such as genetic algorithms [Fal96], operations research methods [CO10], satisfiability techniques [GP10], or constraint programming [Sch09, Sha04]. There are many known bounds on the optimal number of bins which can be used in most of the techniques mentioned above [dC98, LLM03, MT90b].

To illustrate the modeling step, we formulate the bin packing problem as a combina-

torial optimisation problem. Let $I = \{1, \ldots, n\}$ be the set of indices over the n items and $J = \{1, \ldots, m\}$ be the indices of the set of m bins.

$$\min \sum_{j \in J} y_j \tag{2.1}$$

$$s.t. \sum_{j \in J} x_{ij} = 1 \qquad \forall i \in I \tag{2.2}$$

$$\sum_{i \in I} x_{ij} \times s_i \leq C \times y_j \qquad \forall j \in J \tag{2.3}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in I \quad \forall j \in J \tag{2.4}$$

$$y_j \in \{0, 1\} \qquad \forall j \in J \tag{2.5}$$

This formulation uses two sets of decision variables (i.e. (2.6) and (2.7)). We explain the semantics of these variables in the following. The set of x_{ij} variables encodes the assignments.

$$\forall i, j \in I \times J \ : \ x_{ij} = \begin{cases} 1 & \text{if item } i \text{ is hosted in bin } j \\ 0 & \text{otherwise.} \end{cases} \tag{2.6}$$

Naturally, in this formulation there are $O(n^2)$ x variables with n being the number of items considered in the problem. The y_j variables are encoding whether or not a bin j is used. A bin is said to be used if it is hosting at least one item.

$$\forall j \in J \ : \ y_j = \begin{cases} 1 & \text{if bin } j \text{ is hosting at least one item,} \\ 0 & \text{otherwise.} \end{cases} \tag{2.7}$$

The bin packing problem has direct application in the field of data centre optimisation. Workload consolidation usually involves finding an assignment from tasks to servers while ensuring that the total amount of resource required by the set of tasks assigned to a server does not exceed its capacity. The application of constraint programming to this domain has only very recently attracted attention [HDL11, RR11].

Figure 2.1 illustrates an instance solution of the bin packing problem in which we have a set of 8 items with the following sizes: $\{0.6, 0.5, 0.4, 0.4, 0.4, 0.3, 0.2, 0.2\}$. The

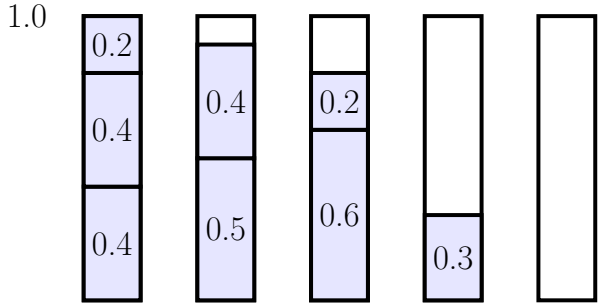


Figure 2.1: Illustrating a solution the bin packing problem.

figure depicts a valid solution to the problem for which items are assigned to four distinct unit-sized bins. Although this solution does not break any constraints in the formulated problem, it is arguably not optimal because items could be packed in three bins as illustrated in Figure 2.2b.

Let the items be numbered as they appear in the list, i.e. $s_1 = 0.6, s_2 = 0.5, \ldots, s_8 = 0.2$. As can be seen on the illustration, this solution fixes variables as follows: $x_{3,1} = x_{4,1} = x_{7,1} = x_{2,2} = x_{5,2} = x_{1,3} = x_{8,3} = x_{6,4} = 1$. On one hand, constraints (2.3) are guaranteeing that the sum of the size of all items assign to the bin does not exceed its capacity, on the other hand, they are ensuring that the relevant y variables are set to 1. Following this example, $y_1 = y_2 = y_3 = y_4 = 1$. Note that any other x_{ij} variables is set to 0 by virtue of constraint (2.3) and the fact that the program minimises the objective function (2.1).

2.2.2 Optimal and Sub-optimal Solutions

Solving constraint optimisation problems usually involves finding a complete assignment from the variables to a value in their respective domains in such a way that all the constraints defined on the problem are satisfied. Such an assignment produces a valid solution for the COP.

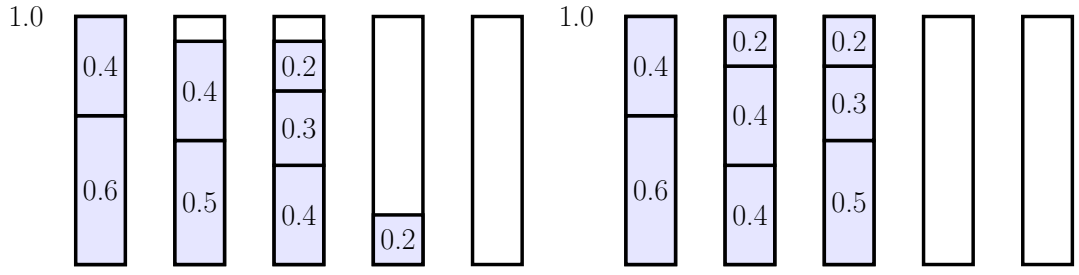
Definition 2. *Solution Optimality in COP. Let $\mathcal{P}$ be an instance of a combinatorial optimisation problem as per Definition 1. Let S and S' be a complete assignment of variables to values from their respective domains such that the constraints of $\mathcal{P}$ are satisfied. The solution S is said optimal if there is no solution S' with a better value of the objective function.*

Although it is in general desirable to find such an optimal solution using complete search, it may not always be possible in reasonable computational time due to the

computational complexity of solving a COP. Proving optimality may need to therefore be abandoned for the sake of producing valid solutions of good quality in a reasonable amount of time using concepts related to heuristic search.

Many ad-hoc heuristic algorithms have been developed for various combinatorial problems to compensate for the NP-hardness of some COPs. This in particularly true in real-world environments in which the time allowed for making decisions can be short because of the operational nature of the decision. In the case of the bin packing problem, a variety of heuristics have been developed including some well-known heuristics such as MAXREST, FIRSTFIT, BESTFIT and NEXTFIT [CGJ97]. Briefly these heuristics operate as follows: MAXREST places the next item into the bin with maximum remaining space capacity; FIRSTFIT places the next item into the first bin that can accommodate it; BESTFIT places the next item into the bin that will have the least remaining capacity once the item has been accommodated by it; finally, NEXTFIT keeps the last bin open and creates a new bin if the next item cannot be accommodated in the current bin, which it will then close.

Using the example instance of BP illustrated on Figure 2.1, Figure 2.2 shows the behaviour of the FIRSTFIT heuristic (Figure 2.2a) in contrast with a provably optimal solution (Figure 2.2b). The FIRSTFIT heuristics fails to return the optimal solution on this particular instance of the bin packing problem. Although its assignment is not optimal, the solution is feasible as all the constraints on the problem on the variables are satisfied. Hence, this solution can be used as an upper bound on the optimal number of bins to be used in this instance.



(a) Solution returned by the FFD heuristic for bin packing. This solution uses 4 bins and hence is not optimal.

(b) Optimal solution using 3 bins to accommodate all the items.

Figure 2.2: Heuristic solution against optimal solution

2.2.3 Online, Semi-online and Offline Optimisation

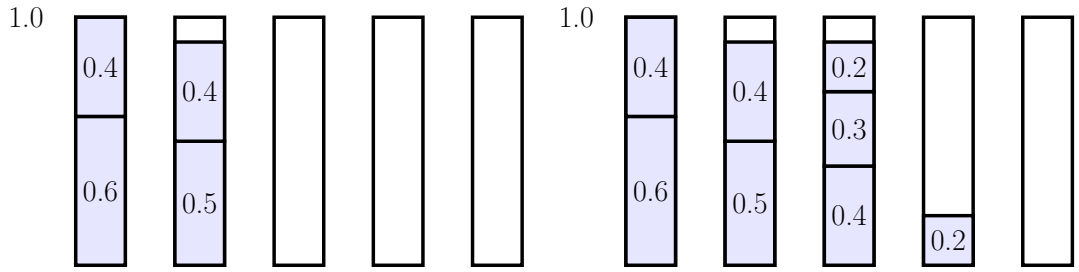
Through this dissertation we tackle various consolidation problems that are expressed in using offline, semi-online or online models. Although, in real production environments, consolidation problems are naturally often expressed as online problems, the offline formulation of these problems are interesting to study for their combinatorial hardness. A combinatorial problem is said to be offline if all relevant information is available at decision time [JW12]. In such a context, provided that the information is accurate, complete solvers can be used to find optimal solutions.

On the other hand, many real-world decision-making situations are naturally thought of as online problems [JW12]. In an online setting, the decision-maker does not have access to all the information necessary to take a fully informed decision. This is due to the fact that some information has yet to be revealed. Traditionally algorithms developed for online settings are studied with concepts developed from competitive analysis [MMS90]. Online optimisation fundamentally differs from stochastic programming in the sense that uncertainties are coming from an unknown future rather than being intrinsic to the objects handled in the problem [Spa03].

Sitting somewhere in between online and offline optimisation is semi-online optimisation. Semi-online optimisation is fundamentally similar to online optimisation in the sense that access to information is limited to what was revealed so far. We distinguish two strategies for decomposing an online problem in a series of offline problems. These strategies handle local information as follows. The main difference resides in the fact that a buffer collects information as time progresses. An optimisation step then takes place considering past information and the information revealed while filling the buffer.

In the case of bin packing, a significant number of articles consider semi-online variants of the problem [BB13, GW95, SY08]. To illustrate semi-online optimisation in the context of bin packing, we come back to the aforementioned example. Let the items arrive in the following order: $L = \{0.6, 0.5, 0.4, 0.4, 0.4, 0.3, 0.2, 0.2\}$. Let us consider an algorithm $OPT(L)$ assigning items in batches L_i in such a way that it minimises the number of bins used after each batch. Let the items arrive in two batches: $L_0 = \{0.6, 0.5, 0.4, 0.4\}$ followed by $L_1 = \{0.4, 0.3, 0.2, 0.2\}$. The algorithm assigns all items in L_0 before having any knowledge of the second batch. In this case, as suggested in Figure 2.3, the algorithm $OPT(L_0)$ returns the assignment pictured in Figure 2.3a. This partial solution uses two bins.

The algorithm is then presented with the second batch L_1 and has to produce a valid



(a) Solution returned after the first optimisation step.

(b) Solution returned after the second optimisation step.

Figure 2.3: Illustration of the semi-online $OPT(L_i)$

assignment without the possibility of moving previously assigned items. Overall, after the second optimisation step, the OPT algorithm has produced a solution using three bins as can be seen on Figure 2.3b. Although valid, this solution is suboptimal compared to a solution produced by an algorithm which has full knowledge of the item list.

2.2.4 Modeling and Solving Constraint Optimisation Problems

Following the modeling step, the model drawn from the application can be implemented using various modeling/solving paradigms. Which paradigm is to be used depends on the context surrounding the application.

Integer Programming (IP) has proved to be a very successful methodology to solve combinatorial optimisation or satisfaction problems [CCZ14]. It differs from Linear Programming by restricting at least one variable to take integer values. The constraints and objective function, if any, are linear expressions. Solving an IP problem is NP-complete. See [NW13] for a survey of linear and mixed integer programming tutorials.

2.2.4.1 Front-end Modeling Tools

With the intention to separate problem modeling from problem solving, various modeling front-ends have been developed. We make use of two front-end modeling tools allowing to conveniently encode the various problems studied in this thesis.

Numberjack[1] is an open source modeling front-end implemented in Python designed to support a number of C/C++ mixed integer solvers such as Gurobi, CPLEX or

[1] http://numberjack.ucc.ie/

SCIP as well as multiple satisfiability solvers such as MiniSat and Walksat and constraint programming solvers. Numberjack provides a layer of abstraction as a unified API for modeling combinatorial problems independently from the underlying solvers [HOO10].

Another tool put to use in the dissertation is developed at Monash University, Australia. MiniZinc implements a free and open-source language for constraint satisfaction and constraint optimisation problem [NSB+07a]. It is rather high-level and solver-independent. It allows for user friendly problem modeling. The model is then compiled in an intermediate language supported by a large number of back-end solvers.[2]

Coming back to the bin packing example, Figure 2.4 and Figure 2.5 show high-level models for the bin packing problem respectively using MiniZinc and Numberjack as front-end modeling tools. We illustrate modeling a bin packing problem with MiniZinc using the instance discussed earlier.

Both modeling frameworks are high level tools used to express the bin packing problem. Their respective first lines are used to declare both constants and variables pertaining to the problem. We typically need to provide the type of the variables we problem is dealing with. In this instance, both x and y variables are encoded using integer values. The set of x variables is encoding the assignment of an item to a bin. The set of y variables is encoding whether a particular bin is used. Then, we state both assignment and capacity constraints that are constraining the problem as described in Section 2.2.1. As a last step, we state that the function that is being minimised accounts for the number of bins used in the solution.

2.2.4.2 Back-end Solvers

Solving NP-hard combinatorial problems in such a way that an optimal solution is found requires the use of a complete solver implementing some form of backtracking system that allows to systematically explore the search space. Many such systems have been implemented using different techniques. In practice, we made use of CPLEX [CPL10], a state-of-the art MIP solver on one side as well as GECODE [Gec06], a well established CP solver.

CPLEX

The CPLEX Optimizer is a commercial suite implementing various mathematical optimisation procedure. It is named after the well known Simplex algorithm

[2] `http://www.minizinc.org/`

```
% A model for the Bin packing problem in MiniZinc.
% Number of items and their capacity
int: m = 8;
int: c = 10;

% set of items
set of int: I = 1..m;

% w[i] is weight of item i
array[I] of int: w = [6, 5, 4, 4, 4, 3, 2, 2];

% set of bins
int: n = m;
set of int: J = 1..n;

% x[i,j] = 1 means item i is in bin j. 0 otherwise
array[I, J] of var 0..1: x;

% y[j] = 1 means bin j contains at least one item
array[J] of var 0..1: y;

% objective is to minimize the number of bins used
var int: obj = sum(j in J) (y[j]);

% Each item must be exactly in one bin
constraint forall(i in I) (
    sum(j in J) ([i,j] == 1)
);

% If bin j is used, it must not be overflowed
forall(j in J) (
    sum(i in I) (w[i] * x[i,j]) <= c * y[j]
);

solve :: int_search(
        [x[i,j] | i in I, j in J] ++ y,
        first_fail, indomain_min, complete)
        minimize obj;
```

Figure 2.4: A Minizinc model for bin packing

for linear continuous optimisation. CPLEX now also implements optimisation algorithms for discrete optimisation [CPL10]. It supports a large variety of languages ranging from C to Python. CPLEX is supported within Numberjack, the modeling front-end for Python.

GECODE

Gecode is a toolkit for developing constraint-based systems and applications. Gecode provides a constraint solver with state-of-the-art performance while being modular and extensible. GECODE has support within Minizinc and implements a number of global constraints.

These solvers will be used as tools to solve combinatorial problems defined and study in the reminder of this thesis.

```
from Numberjack import *

def model_binPacking(data):

    model = Model()

    x = Matrix(data.nItems, data.nBins)
    y = VarArray(data.nBins)

    # Cost of a solution
    obj = Sum(y)
    model.add(Minimise(obj))

    # Constraint assign item
    for j in range(data.nBins):
        model.add(Sum(x.row[j]) == 1)

    for j in range(data.nBins):
        xs = [x[(i, j)] for i in range(data.nItems)]
        coefs = [data.itemSizes[i] for i in range(data.nItems)]
        model += Sum(xs, coefs) < y[j] * data.BinCap

    return (obj, x, y, model)

def solver_binPacking(data, param):
    (obj, x, y, model) = model_binPacking(data)
    solver = model.load(param['solver'])

    solver.solve()

    print "Obj : {0}".format(obj.get_value())
    print "X : {0}".format(x)
    print "Y : {0}".format(y)

class BinPackingData:
    def __init__(self):
        self.itemSizes = [6, 5, 4, 4, 4, 3, 2, 2]
        self.nItems = len(self.itemSizes)
        self.nBins  = self.nItems
        self.BinCap = 10

solver_binPacking(BinPackingData(), input({'solver':'CPLEX'}))
```

Figure 2.5: A Numberjack model for bin packing

2.3 Optimisation Problems in Cloud Computing Systems

Over the last decade cloud-based services have attracted a lot of interest from the research community. We review work carried in an effort to maintain control over the cost of owning and operating cloud systems. A particular focus is put on workload management techniques as they pertain to this dissertation.

2.3.1 Cost Models for Data Centres

Efforts to optimise cost aspects of cloud computing have covered many aspects of cloud systems. A cost of ownership breakdown of a cloud structure can be found in [GHMP08, DWC10]. The cost analysis suggests that there are many opportunities for optimisation approaches to reduce it. An economic focused approach of such a breakdown can be found in [SS10] and [CDP05] in which authors show the relative cost of various key elements needed to build, maintain and operate a cloud system. In particular, energy cost models are found to be widely diverse and highly dependent on technologies implemented in and across the data centres.

Because of the rapidly changing technologies, a precise quantification of the budget needed to develop a cloud system is difficult to estimate. It is generally reported in the literature, see [KBSW11, MBS+11], that the computational equipment (i.e. servers, CPUs, memory and storage equipment) represents roughly 45% of the overall cost. The infrastructure and networking facilities needed to operate these servers are accounting for up 40% of the cost. Finally, the electricity bill represents 10 to 15% of the operational cost of running such a structure.

A very comprehensive survey [DWF16] published in 2016 discusses the recent contributions on energy consumption modeling in data centres and cloud system. The survey covers a wide range of facets relevant to cloud infrastructures spanning from low level models for digital circuitry energy efficiency to high-level workload management procedures. The authors review more than 200 models relevant for the purpose of understanding and optimising such systems. These models are systematically classified according to a taxonomy proposed in the survey. The work presented in this dissertation falls in the scope of rather high-level software-centric workload consolidation policies.

In the extensive amount of research carried by the research community focused on cloud optimisation over the last decade, much effort has been invested in developing comprehensive and realistic models describing cloud systems [GHMP09]. It has been pointed out numerous times that beyond the cost of building and maintaining such systems, how assets are utilised to accomodate the demand for computation is a critical aspect. Indeed, appropriately managing workloads has been shown to be a key challenge for building sustainable cloud systems [BB10b, GHMP08, Koo08]. In this context, this dissertation puts the focus on workload management.

2.3.2 Managing Workloads in Data Centres

During normal operations, cloud providers face a set of tasks to be assigned to physical machines in the data centre. Tasks usually take the form of virtualised workloads that can run concurrently on a physical machine [BB10b]. We discuss further details of virtualisation technologies in the next section. Data centres are typically over-provisioned in terms of number of available machines to be able to cope with high fluctuations in the demand for computing power [AFG+09]. Having a much larger pool of machines than needed at most times also allows one to design fault resilient systems [GHMP08]. The downside of over-provisioning is that only a small fraction (6-12%) of the electricity used by data centres can be attributed to productive computation. [3]

2.3.3 Workload Consolidation and Virtualisation Technologies

The aim of workload consolidation, usually implemented through virtualisation of assets, is to increase the utilisation of a subset of servers. Consolidation is typically achieved by allocating multiple computational tasks on the same physical machine [AGH+15]. In turn, workload consolidation allows data centre operators to spread workload over a smaller set of machines so that those remaining unused can be either powered down or put into a standby mode. Ultimately, consolidation techniques increase the overall resource utilisation of servers actually used for computation and helps to reduce the overall number of servers required by the cloud provider to accomodate the workload.

Consolidation techniques can be implemented in various ways depending on the infrastructure's underlying capabilities and technologies, but also on the focus and needs of the applications running on such a system [AGH+15, CPW07, GHZ13]. Virtualisation of computing environments is a key technology to help develop workload consolidation mechanisms. This is particularly true while implementing dynamic consolidation schemes as they require the virtualised computing environment to be migrated from a host machine to another one [FNCR11].

A number of maturing technologies allowed for the development of layers of abstraction between computing environment and computing hardware. Virtual Machines (VMs) are an example of such an abstraction by implementing a software layer emulating a computer's physical hardware. VMs are an essential mechanism to implement

[3] `http://www.sallan.org/pdf-docs/McKinsey_Data_Center_Efficiency.pdf`

workload consolidation policies in the sense that they allow multiple fully virtualised environments to be run concurrently on the same computing hardware. A survey on VM consolidation and green cloud computing can be found in [HI15].

Workload consolidation policies can be either static or dynamic [BKB07]. *Static workload consolidation* policies usually use per-VM historical or estimated average/peak resource demands to decide on an assignment. Assignments of tasks to servers may not be recomputed for long periods of time. Depending on the kind of workload and the service offered by the cloud provider, a task may be assigned to a machine for its entire duration without ever be moved to an other host.

In contrast, *dynamic workload management* is implemented on short timescales, preferably shorter than periods of significant variability of the resource demand [BB12]. It is a reactive approach in which servers are continuously monitored and the reconfiguration of VMs to servers is triggered when the servers are either overloaded, under-utilised or for some reason are not available for hosting workload. Many data centres have implemented the necessary technologies and infrastructure for workload migration.

There are several reasons for migrating the load of one or more virtual applications from their current host servers to different ones. For example, if the load on a server is very high (or very low), or if the server is about to shut down, then one might want to move some or all the virtual machines from that server to an other. Also, if there is a server where the energy cost per unit of computation is cheaper, then one might want to reassign some virtual applications to that server so that the overall cost of energy consumption is reduced. More specifically, the challenge of dynamic workload consolidation is to consolidate server workload efficiently by deciding which virtual machine to migrate, where to migrate, when to migrate, and, when and which servers to switch on or off. Various objective functions coul be considered, so that the overall energy costs are minimised [GHZ13].

2.3.4 Workload Consolidation as a Packing Problem

A natural candidate modeling framework for workload consolidation problems in cloud systems is the family of combinatorial packing problems [RKS+08]. This family of problems can model many workload consolidation problems quite naturally since its core is to find a complete assignment of smaller items to larger containers (traditionally referred to as bins) under capacity constraints. Using cloud computing terminology, pieces of virtualised workloads can be regarded as items to be packed on servers (bins)

in such a way that one does not exceed the bin's capacity. Many workload management models are to some extent inspired by variations of the bin packing problem where available resources on servers are seen as bin capacities and tasks seen as items to be assigned [GHZ13, LTC14b, WTAPB15, HDL11, RKS+08].

As discussed in Section 2.2, in classical bin packing settings we are given a set of items along with their sizes and a set of bins with unit capacity. In the optimisation formulation, the objective is to find an assignment from items to bins such that the total number of bins used to accommodate items is minimised. See [CJCG+13b] for a comprehensive review on packing problems and algorithms to solve them. Many extension and variants of bin packing are relevant to this dissertation. In particular, the Variable-Size bin packing problem (VBP) is a variant in which bins have different capacities and the problem is to minimise the sum of the wasted space over all used bins [FL86]. This model fails to capture the notion of item's time to live in the bins which is a crucial aspect of any workload consolidation problem. Another variant of bin packing related to packing VMs for consolidation purposes is the Vector Bin Packing problem [CK04]. In this variant, one considers multiple dimensional objects to be packing in containers. This models are usually not modeling items with duration.

Some bin packing problem variants, usually classified in the family of Dynamic bin packing problems, capture the notion of items having a lifespan in bins [BJK14]. These models typically capture the latter by adding a *duration* attribute to items. Items, once assigned to a bin, will be consuming space (resources) over the time of their lifespan. In this framework, items are typically characterised by size, arbitrary arrival and departure times (or duration). The objective is to minimise the maximum number of bins ever used over time. Although closely related, in subsequent chapters we study the problem from different angle where we aim to minimise the cumulative cost resulting from using bins over a given horizon.

As a related extension, the fully dynamic bin packing problem [IL09, BJK14] allows to rearrange the items across the bins to retain a minimal number of used bins. Multiple contrasting models are discussed in the literature. Differences revolving in particular around the notion of item reassignments. For instance, the case in which an arbitrary upper bound (at most k) on the number of item rearrangements allowed in an optimisation step in discussed in [JK13]. This upper bound on the number of rearrangements is expressed through a ratio computed as the number of items reassigned over number of items arriving. On the other hand, some models are integrating a cost linked to rearrangements within the objective function [CMOS13a]. This rearrangement of items between bins is known to the cloud computing community as task or VM migrations.

2.3.5 Extracting Evaluation Datasets

A significant proportion of the work carried out in this dissertation analyses workload management strategies in different contexts. Beyond synthetic data sets, we are interested to study how these policies behave facing workloads coming from a real world data centre. Unlike more traditional scientific or high-end computing environments, cloud computing serves a much broader variety of workload profiles. For instance, long-running Internet services, large-scale data analysis or even testing and developing of software applications [RTG+12, DKC13] have been brought to cloud environments.

In 2011, Google data centre engineers released a substantial data log tracing the activity of one of their data centres over the period of 29 days [RWH11]. We present here an analysis of this data sets as it illustrates relevant aspects of workload as faced by cloud providers. The information extracted from this dataset will be used to evaluate consolidation policies throughout this dissertation.

The trace contains information about the computing power hosted locally and the workload to be processed on it. To a large extent the data set illustrates well the unique challenges that cloud providers have to face. The trace includes information on millions of tasks scheduled across 12,583 machines. The trace consists of more than $40,000$ cloud applications, which are called numerous times by thousands of users in the form of jobs [DKC13]. In this section, we review the first 48 hours of the Google trace.

2.3.5.1 Data Extraction

In the trace, a task represents a Linux program, possibly consisting of multiple processes, to be run on a single machine. The trace considers a set of resources: CPU, RAM, disk space, disk-time fraction (I/O seconds) along with the resource capacities for each machine, the user-specified maximum requirements of resources, and time-variable resource usages for all tasks. These values have been normalized between 0 and 1, according to the maximum capacity of a resource from the entire set of machines. Moreover, the trace includes tasks' status (refered to as *events*), such as: *submitted*, *waiting to be scheduled*, *running*, *evicted*, *killed*, *resubmitted* or *finished*. The reader can refer to [RWH11] for a complete discussion of the original trace.

To evaluate workload consolidation policies, we are interested in extracting information on tasks fully processed by the cloud system. We consider a task in the trace completed if it ran till completion, or was evicted and not submitted again. Tasks that

do not meet this condition in the trace were discarded, since the information we have is too little to properly compute their properties. We then define the task's duration as the time difference between the task's completion and its last submission time. The application described in the trace considers that tasks can not be partially completed and resumed later. Furthermore, we also extracted the task resource utilisation profiles of CPU and RAM over its duration. We use the *jobID* as the unique identifier for a job, and for each of these jobs we extracted the duration of all its tasks, the user-specified maximum requirement for each resource, and a set of actual usages for each resource for all of its tasks. Moreover, we did not use all the attributes available from the trace. We focus mainly on the variables that described the job events, tasks events, tasks usage, and machines attributes.

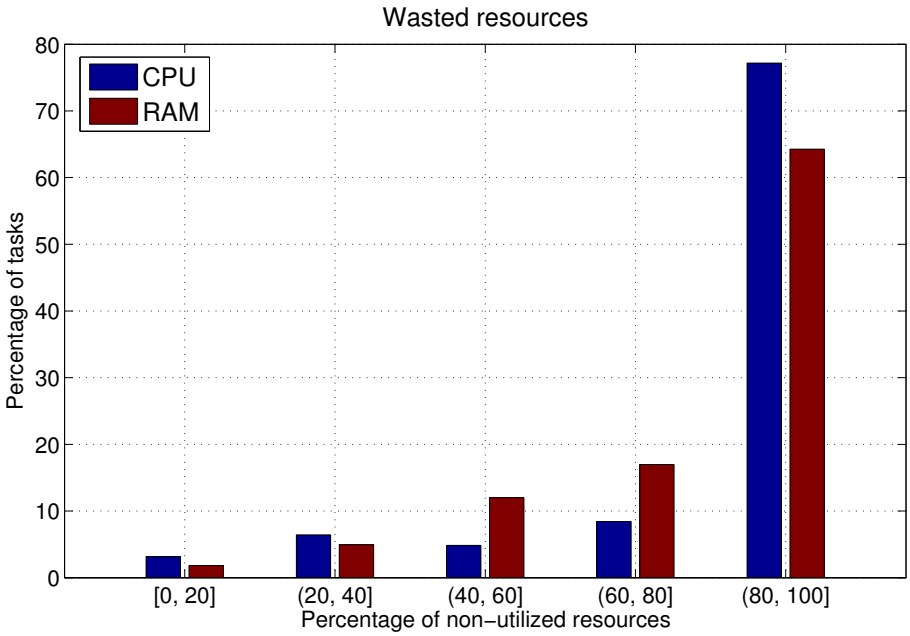


Figure 2.6: An analysis of the percentage of tasks with non-utilized resources. The percentage of non-utilized resources is computed as the difference between the allocated and the utilized resources by a task.

In this dissertation, when use is made of this data set, we either consider one resource (i.e. CPU), or a combination of resources (i.e. CPU and RAM), since these were the only two resources for which one could retrieve the relevant information for all machines and tasks. The attributes that we considered for CPU and RAM are: the CPU rate, which indicates the average CPU utilisation for a sample period of 5 minutes, and the canonical RAM usage, which represents the average RAM consumption for the same sampling period. Upon submission of tasks, the cloud system users are invited to provide an estimation of the CPU and RAM needed. We refer to this quantity as the users' resource limits. This information, although an approximation, is in turn used by

the cloud provider to decide on an assignment for any given task.

2.3.5.2 Characterising the Workload

For the studied tasks, we noticed that 2.8% of the tasks had a duration of less than 0 seconds, meaning that they were submitted but were never scheduled. These tasks were naturally discarded from the data set. Moreover, tasks with a CPU consumption higher than 1.0 were also removed from the experiments, since those values could not be compared directly to the machines' resources. These only occurred for 13 out of $\sim$ 1.8 million tasks and are referred to as anomalies by the team that released the original trace.

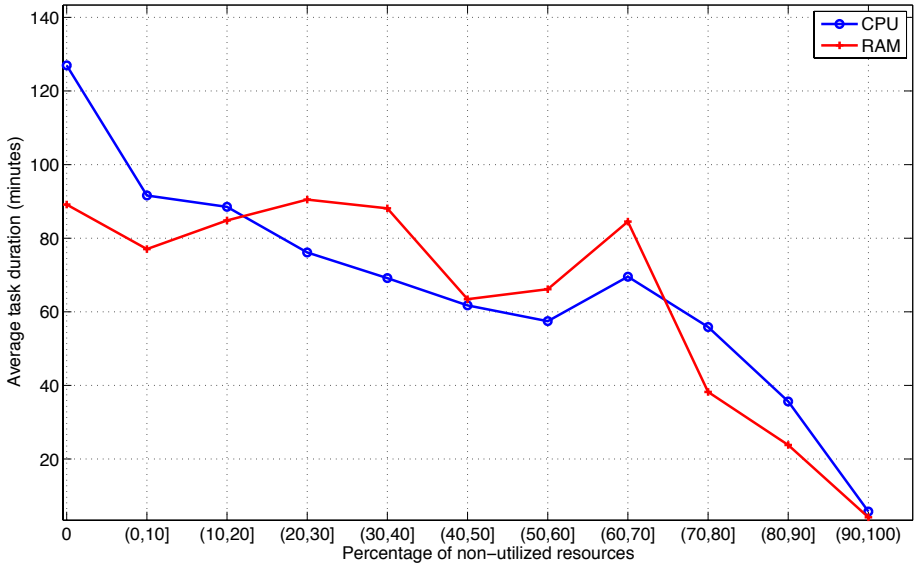


Figure 2.7: Relationship between over-requested resources and the duration of the tasks

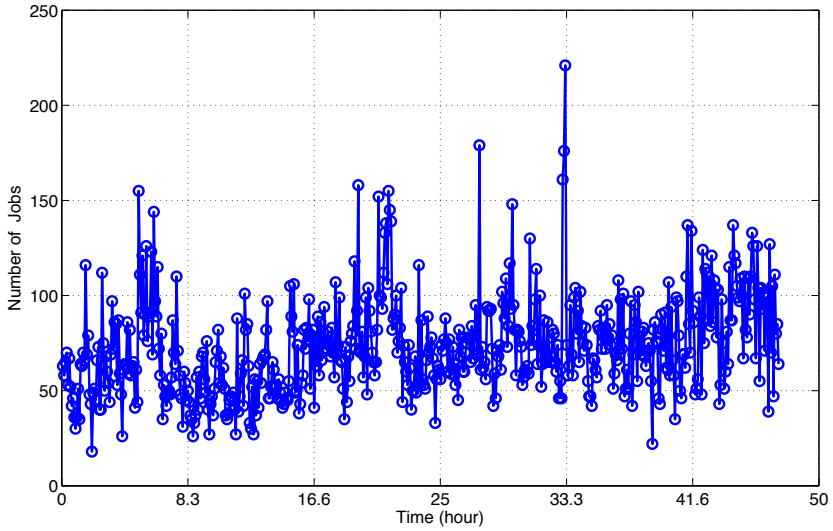
We investigated whether there was a significant difference between the users' resource limits and the tasks' maximum resource consumption within a 48 hours time frame. Figure 2.6 presents the tasks that requested unnecessary resources. We observed that the majority of tasks, namely 77.4% in the case of CPU and 63.7% in the case of RAM, requested between 80% and 99% more resources than what the task consumes at maximum during its execution. Nevertheless, it is important to emphasize that these errors occurred mostly for tasks with low duration (i.e. less than an hour) and the number of tasks with low duration are significantly more than those of longer duration. Figure 2.7 presents the relationship between tasks' duration and the percentage of the tasks' non-utilized resources, we divided the percentage of non-utilized resources into buckets of

10% each. Figure 2.7 shows a clear trend in which tasks with long duration were the ones that requested between 0% and 10% more resources than necessary, while the tasks with shorter duration requested at least 70% more resources than necessary. For instance, the figure shows that tasks that requested at least 90% more resources than necessary, have an average duration of 5.7 minutes when the resource considered is CPU and of 4.1 minutes when the resource considered is RAM. On the other hand, the tasks that requested between 0% and 10% more resources than needed, have an average duration of 91.6 minutes when the resource considered is CPU and of 77.1 minutes when the resource considered is RAM. Furthermore, for both resources the absolute difference between the user's resource limit and the maximum utilized by a task is rarely large. For instance, only 3% of the tasks requested more than 0.05 CPU units than needed and less than 2% of the tasks requested more than 0.05 RAM units than needed.

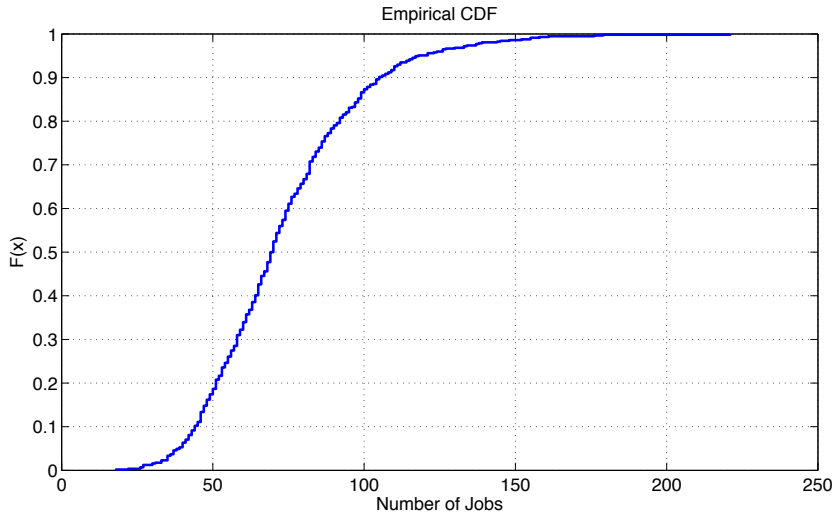
Moreover, we discovered that 17.4% of the tasks requested less CPU than needed. In the case of RAM, the number is much lower. We observed that only 1.9% of the tasks requested less RAM than needed. In addition, we see that 96.7% of the tasks that exceeded their CPU limit were tasks that requested between 0.0 and 0.05 of CPU. Similarly, 95% of the tasks that exceeded their RAM limit were tasks with a limit between 0 and 0.064 of RAM. This is due to some internal policies in the Google cluster in which users can request less than what they utilize without beeing evicted [RWH11]. Since requesting 0.0 units of CPU will result in an obvious underestimation, we replace 0 with the next minimum CPU request, 6.247e-7. Similarly, we replaced 0.0 RAM limit with the next minimum value, i.e. 9.53e-7.

Figure 2.8a presents job arrival rate. The x-axis presents the time between 0 hours and 48 hours, namely the entire time frame that was studied here. The y-axis presents the total number of jobs submissions. Each point represents cardinality of the jobs submitted within a 5 minute window. The figure shows a moderate fluctuation in terms of incoming jobs rate, where values range from 18 to 221 jobs. Moreover, Figure 2.8b presents the CDF of the values from Figure 2.8a. The figure shows that almost 90% of the time the total number of jobs that will arrive in 5 minute will be at most 100. Furthermore, it is expected that, 67% of the time between 50 and 100 jobs will arrive in 5 minute, 20% of the time it is expected that less than 50 jobs will arrive, and 13% of the time it is expected that more than 100 jobs will arrive.

Figure 2.9 presents the number of tasks associated to each job over the 48 hours studied. It is clear from the figure that the majority of the jobs are composed of a small number of tasks. We are considering the distribution of number of tasks per job and



(a) Job arrival count in the 48 hours studied.



(b) Job arrival CDF.

Figure 2.8: Characterising jobs arrival rate

discovered that 76% of the jobs consist of a single task, 16% of jobs are composed from 2 to 50 tasks, and only 8% of the jobs are composed by more than 50 tasks (up to 10,500 tasks in rare occasions). Other studies (see [RTG+12, DKC12]) suggest that the patterns of utilisation in the trace indicates that the cluster is used by several organizational entities, each executing its own particular mix of tasks. This type of environment is usually characterized by a large dynamic range of resource demands with high variation over short time intervals. Reiss et al. [RTG+12], studied the job inter-arrival rate for the whole trace and discovered that around 40% of submissions recorded less than 10 milliseconds after the previous submission, and that the median

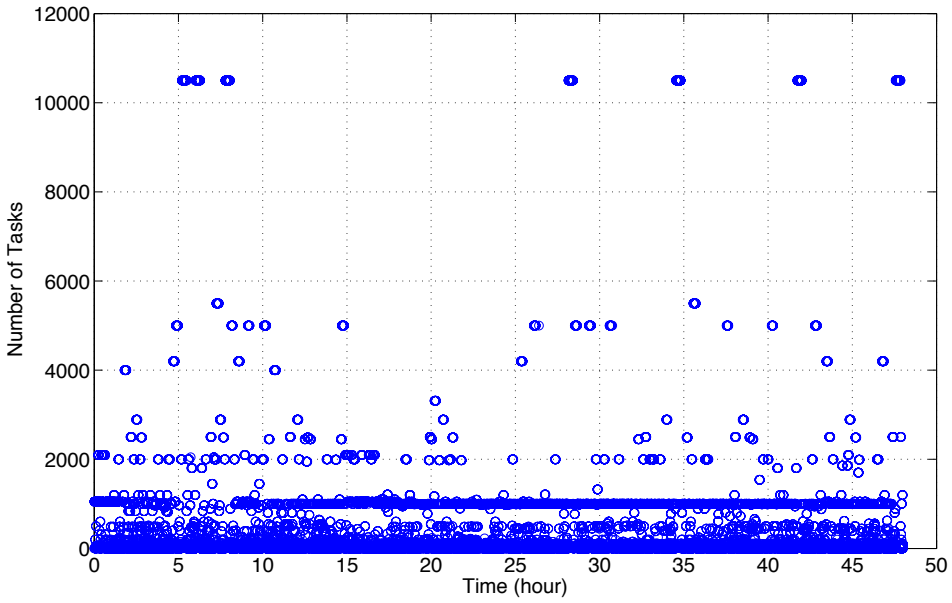


Figure 2.9: Tasks count per job

inter-arrival period is 900 milliseconds.

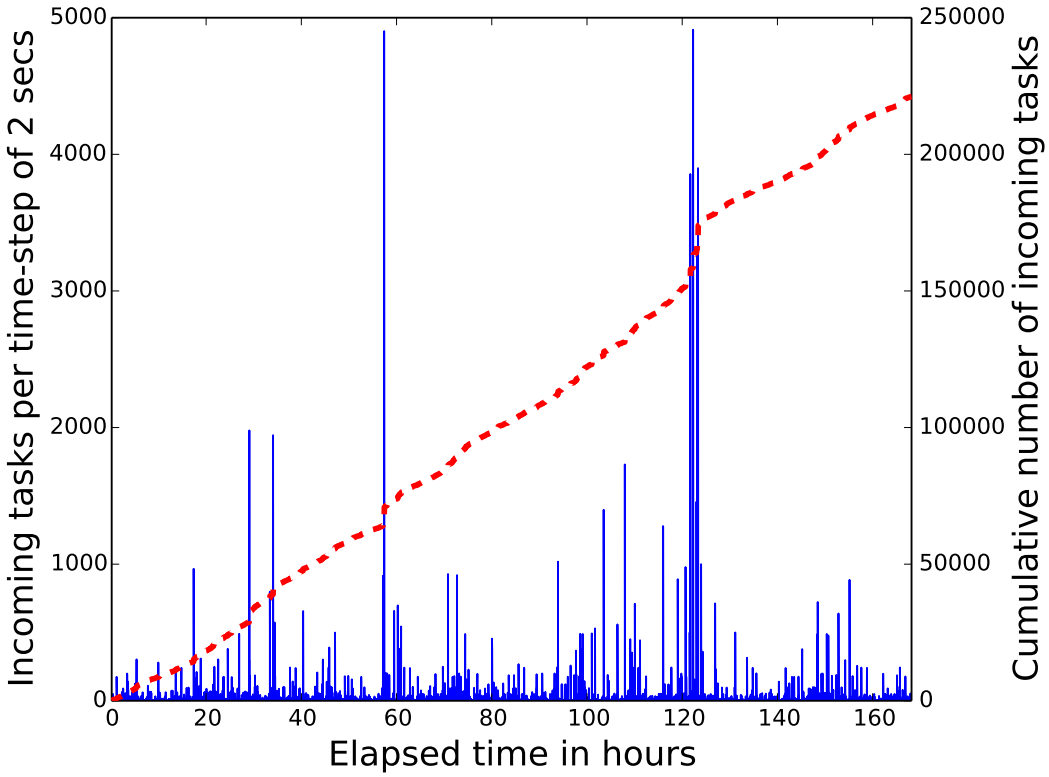
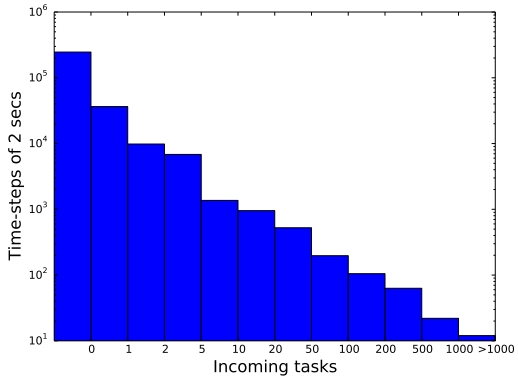


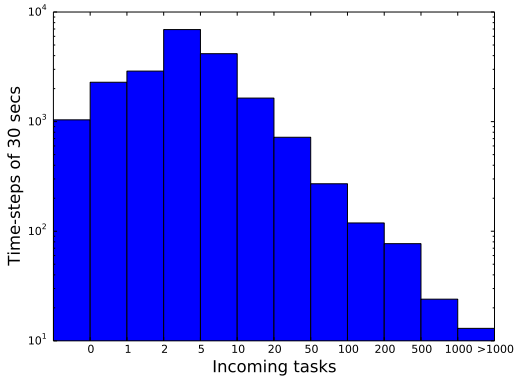
Figure 2.10: Number of incoming tasks over elapsed time

Figure 2.10 shows the high variability in the number of incoming tasks received over

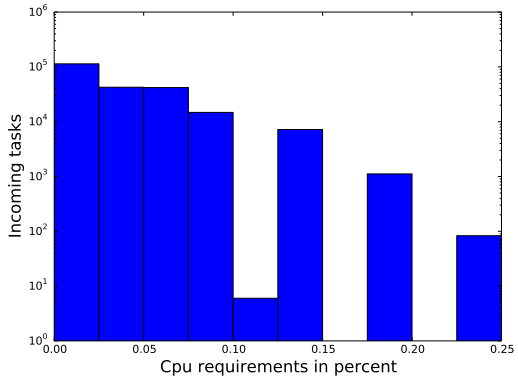
the time. Each vertical bar represents the number of incoming tasks (left y-axis) received within a time period of 2 seconds at a specific time (x-axis). The cumulative number of incoming tasks (right y-axis) is depicted by the red dotted line. In the dataset, the number of tasks received by two consecutive time steps can differ by several orders of magnitude. There are two noticeable peaks of around 5000 tasks arriving after 57 hours and 123 hours. The second noticeable peak is closely surrounded by other peaks of several thousands of incoming tasks. This specificity in the dataset is of particular interest. It allows us to compare the behaviours of the different policies in case of intensive demand of resources.



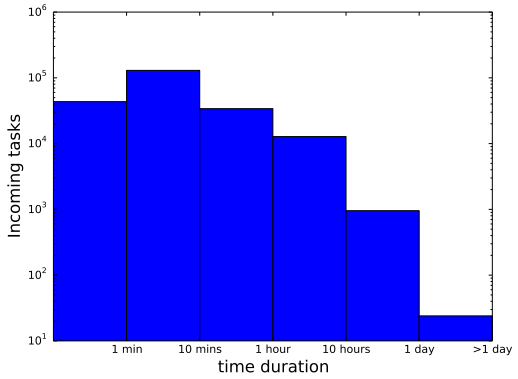
(a) Number of time windows of size 2s (y-axis log scale) presenting x incoming tasks (x-axis)



(b) Number of time windows of size 30s (y-axis log scale) presenting x incoming tasks (x-axis)



(c) Number of incoming tasks (y-axis log scale) per task cpu requirements (x-axis)



(d) Number of incoming tasks (y-axis log scale) per task durations (x-axis)

Figure 2.11: Distributions of incoming tasks in the dataset

Figure 2.11 shows various task distributions of the dataset according to different parameters. Figure 2.11a, respectively, Figure 2.11b, shows the number of time steps of 2s, resp. 30s, when varying the interval of tasks received within a time step. For 2s and 30s, the number of time steps decreases in a logarithmic manner as the number

of incoming tasks increases. Over the week of incoming tasks, for the 2s time period, more than 246000 time steps of 2s representing a sum of 56 hours, receive no tasks (Figure 2.11a), while only 1038 time steps of 30s, representing a sum of twenty one hours, are empty (Figure 2.11b). The difference between the number of time steps of 2s and 30s remains significantly high when one or two tasks are received within a time step. Then, as expected, since tasks have more chance to be pooled within time steps of longer time periods, there are more time steps of 30s that receive between six to ten tasks, eleven to twenty tasks, ..., > 1000 tasks than the number of time steps of 2s. Note that, for both distribution 2s and 30s, there are more than 700 time-steps receiving between twenty and a hundred tasks. Finding an optimal placement for these time steps remains challenging even for state of the art techniques and solver [CMO16]. From more than a hundred tasks received during a time step, the difference between the number of time steps of 2s and 30s is tightening and becomes almost equal for of tasks> 1000 tasks. There are thirteen time steps of 30s against twelve time steps of 2s receiving more than 1000 tasks. This pattern is specific to the data-set built from a real trace of incoming tasks. Peaks of incoming tasks are spread across time. In this last case, the associated placement problems are very hard to solve optimally within a service level agreement matching on-demand QoS expectations.

In Figure 2.11c, the incoming tasks are sorted by percentage of CPU-requirement of the largest capacity machine, which represents a standard machine in our experiments. The CPU-requirement of each individual task does not exceed 25%. The number of incoming tasks decreases in a logarithmic manner as the CPU requirement increases. Most of the tasks in the data sets require less than 7.5% of the CPU capacity of a standard machine. In this case, a machine can host more than ten tasks.

Figure 2.11d shows the distribution of incoming tasks per duration as reported in the trace. The dataset contains a wide range of task durations. It is characterized by a majority of tasks (>170000) finishing before ten minutes. There are around 40000 tasks lasting between ten minutes and one hour. The remaining tasks, approximatively 15000, last more than one hour.

2.4 Workload Management in Geographically Distributed Clouds

In this section, we consider a case-study introducing fundamental aspects of workload management in distributed cloud systems. This section is used as a vehicle for explain-

ing the behaviour of Internet data centres (DCs) under several typical settings. We aim to highlight the importance of properly handling workloads in a setting where electricity can be sourced on different markets. We show how to make use of optimisation techniques coupled with statistical learning to appropriately handle workloads in cloud structures.

The key idea here is to take advantage of price differentials in a multi-electricity market setting. We present a methodology for studying the energy cost implications of minimising data centre energy costs under different operational and energy cost prediction regimes. We then systematically study the impact of the level of price variability, time lag between locations due to the geographical distribution, reconfiguration delay, and accuracy of price predictions on the overall electricity cost associated with managing a network of data centres.

For various operational and strategic reasons, such as speed and latency, redundancy of both equipment and data, networks of data centres are sometimes structured in a geographically distributed fashion [GHMP08, ZCB10]. The cost per unit of computation can vary significantly between various locations due to regional specificities [QWB+09]. Noticeable efforts motivated by the importance of energy costs in operating a DC have been made to take advantage of these price differentials [LLRL12, RLXL10, SLX10, BGG+10] and to design energy-aware routing protocols [QLM12].

From a combinatorial optimisation point of view, we can formulate the problem of managing a data centre as an assignment problem where one tries to allocate workloads to a set of data centres such that an overall energy cost function in minimised. This cost function should be a function of the various characteristics of the set of data centres and the forecasted electricity price at each location. Of course, factors such as geographical spread and the time needed to reconfigure the system should also be incorporated into the assignment problem.

We present an approach to simulating realistic electricity prices using a time-series analysis technique. We aim to capture the generic behavior of electricity prices on a wholesale electricity market. We will also introduce a simple approach to simulating Gaussian errors while predicting prices over a short time-horizon. The focus of this section is put on a variety of problems of interest in data centre management.

2.4.1 Models for Electricity Prices and Price Prediction Errors

In order to study the electricity costs associated with managing a data centre (DC), a realistic model for electricity price dynamics on a wholesale market is required. Figure 2.12a shows the weekly dynamics of the actual spot price of electricity on the Irish market over the first seven days of 2009 at 30 minutes intervals. Electricity price is naturally studied as a time-series in which each data point describes the price of electricity at a specific moment in time.

There are various approaches to building predictive models for real-time market electricity prices, for example GARCH models [HLLz05], wavelet models [TZWX10], artificial neural networks [YSL04, HPS01], and other machine learning-based methods [IOS12a]. In this case-study, we used a time-series analysis procedure referred to as the Box-Jenkins (BJ) method [BJ70] to build an Auto-Regressive Integrated Moving Average (ARIMA) model characterizing the electricity price behavior over a day. A complete description of techniques used to model time series data can be found in [CC09]. The ARIMA model is a commonly used tool to understand, model and predict future values of a time-series P_t (see [CENC03, RB]).

The ARIMA models the behavior of a time series using two components. The first one is the autoregressive (AR) part,

$$AR(p) : P_t = \sum_{i=1}^{p} \varphi_i P_{t-i} + \epsilon_t$$

which states that values of the series are partially determined by its past values. The second part is the moving average (MA) part,

$$MA(q) : P_t = \epsilon_t + \sum_{j=1}^{q} \theta_j \epsilon_{t-j}.$$

Combining these two components the model

$$ARMA(p,q) : P_t = \epsilon_t + \sum_{i=1}^{p} \varphi_i P_{t-i} + \sum_{j=1}^{q} \theta_i \epsilon_{t-j}$$

is a powerful tool that builds a rather simple model for a time-series. Note that ϵ_t is a set of independent variables identically distributed according to a Gaussian distribution $N(0, \sigma^2_{Price})$ with σ^2_{Price} being a measure of the variability of the price.

In order to grasp the seasonal nature of the series we used the generalized SARIMA$(p,d,q)(P,D,Q)[s]$ model, where d and D are, respectively, the order of

the ordinary and the seasonal differentiation, p and P are the orders of ordinary and seasonal AR processes, and q and Q the orders of the MA processes. Finally s is the frequency of the seasonality, which is 48 in our case representing a period of 24 hours at a fidelity of 30 minutes.

The Box-Jenkins approach is a process that iterates over a set of candidate ARIMA models to find the best fit of a time-series to its past values. Figure 2.12 shows the various steps that were undertaken to produce an accurate model for electricity price. The procedure aims at finding best fit for parameters φ_i and θ_i to the data. The first step is to ensure that the time-series under study is stationary, i.e. that the mean and variance over time is constant, and that we accurately model the seasonal effect, if any.

The actual series as seen in Figure 2.12a does not fulfill the stationary property, and shows a clear seasonality over a range of 48 time periods. Thus, the series was differentiated twice including a seasonal differentiation:

$$\bigtriangledown_{48} dP_t = dP_t - dP_{t-48}$$

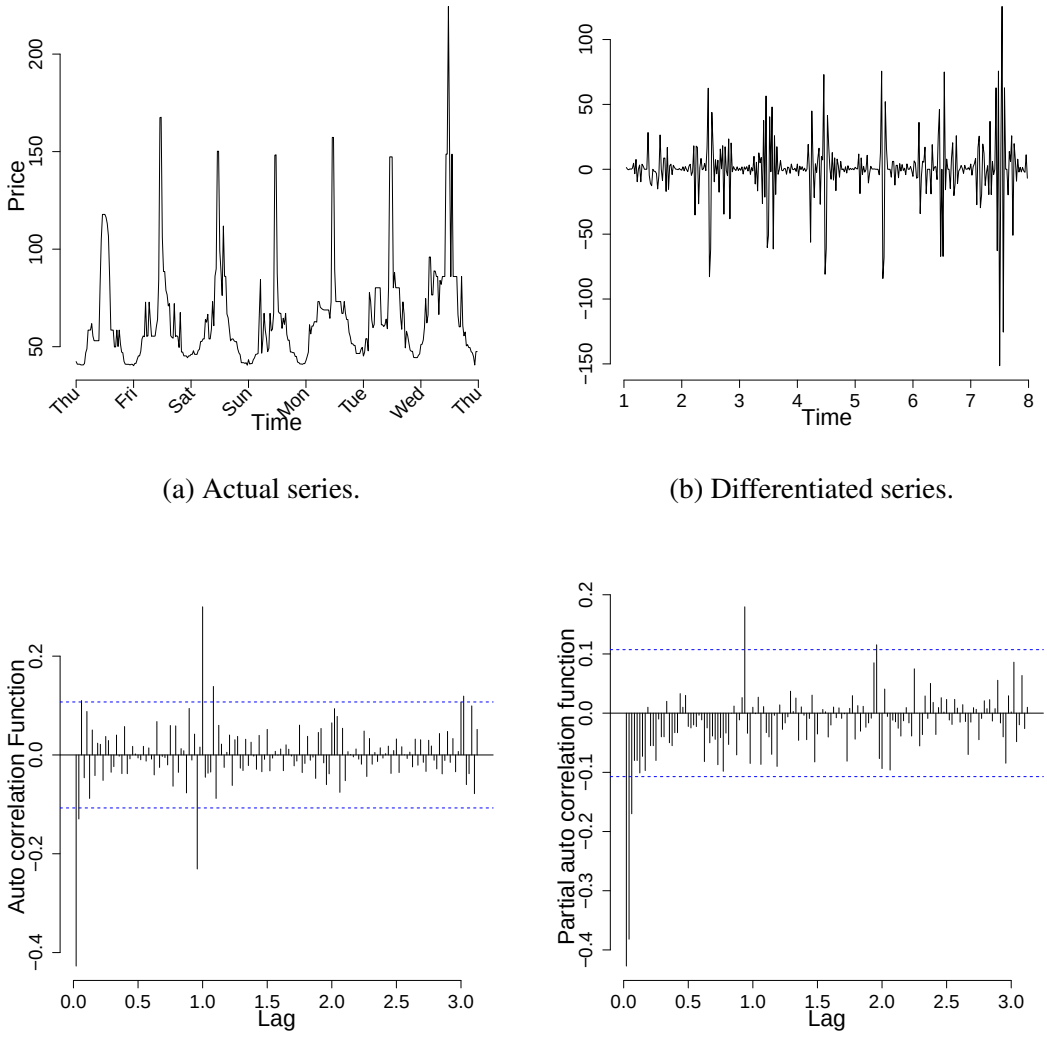
with

$$dP_t = \bigtriangledown P_t = P_t - P_{t-1}$$

As a result, Figure 2.12b shows that a stationary series was achieved by differentiating the time-series twice thus fixing the orders $d = 1$ for the regular and $D = 1$ for the periodic component.

The next step allows us to define the orders of both the AR and MA processes, respectively p and q in our model. For doing so, we refer to the Autocorrelation Function (ACF), illustrated in Figure 2.12c, and the Partial Autocorrelation Function (PACF), presented in Figure 2.12d, of the differentiated series. The quick decay of values on the ACF suggests an $AR(p)$ process. The value of p should be read on the PACF as the last value significantly different from 0. Hence $p = 3$. Similarly, the order of the MA process is read on the ACF. Hence $q = 3$. Then, the same procedure is repeated for the seasonal component by taking into account a 48 period lag allowing us to fix $P = 1$ and $Q = 2$. Values of best fit of parameters θ_i and φ_i are listed in Table 2.1. The last step in the BJ method is to check that the residuals of the model are showing white noise properties.

Figure 2.13 illustrates the range of situations that one can simulate by varying the σ^2_{Price} parameter. This parameter affects the amount of noise that the generated series shows against the theoretical ARIMA model. As the figure shows, even with highly fluctuating prices (e.g. $\sigma^2_{Price} = 500$) the trend over a day still appear. In the following,



(a) Actual series.

(b) Differentiated series.

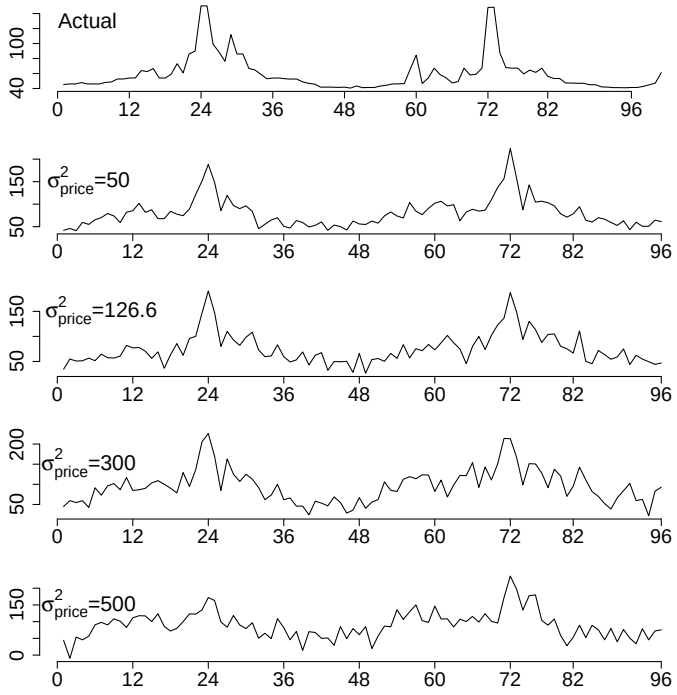
(c) Autocorrelation function.

(d) Partial auto correlation function

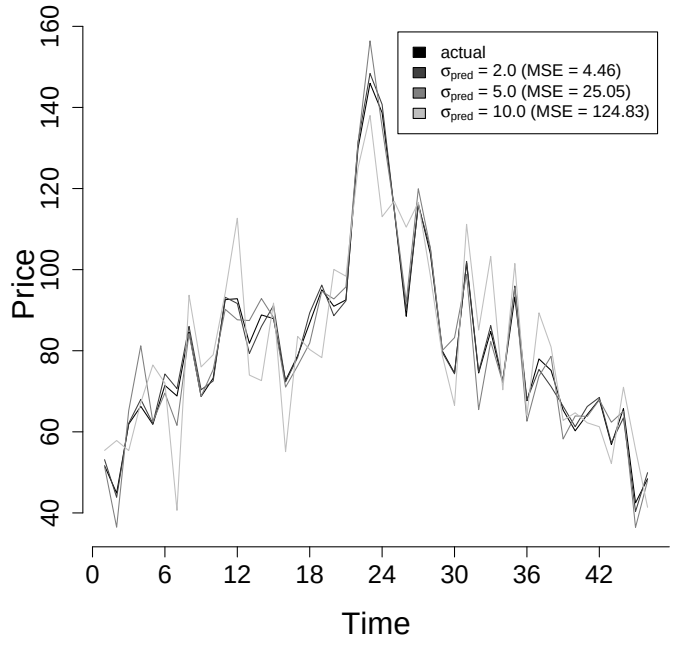
Figure 2.12: Modeling electricity prices. An iteration of the Box-Jenkins method.

we will use σ^2_{Price} to carry experiments in which price series are simulated with a controlled intrinsic variability.

In addition to a model for electricity prices, we aim to provide an empirical insight into the impact of price prediction errors on the overall cost of the optimal assignment of workload in an data centre. To this end, we modeled forecasting errors for a particular time slot t as being distributed according to a Gaussian distribution centered on the actual value. Hence, let $P = \{p_1, \ldots, p_T\}$ be a price series and $\hat{P} = \{\hat{p}_1, \ldots, \hat{p}_T\}$ be a simulation of its forecasted values. Given P, we can build $\hat{P}$ such that $\hat{P}_t = \mathcal{N}(P_t, \sigma_{pred}), \forall t \in T$. Here σ_{pred} describes the standard deviation of the predictions.



(a) Simulated price series.



(b) Simulated forecast errors.

Figure 2.13: The upper plot shows a week's price dynamics seen on the Irish electricity market. The other plots represent several simulations ranging from a clean to a noisy trend. Simulations were generated on 96 time periods with $\sigma^2_{Price} \in \{50, 126.6, 300, 500\}$. Below, simulated forecast errors: $\sigma_{pred} \in \{2.0, 5.0, 10.0\}$ ranging from good predictions to inaccurate ones.

Table 2.1: Seasonal ARIMA(3,1,3)(1,1,2)[48] model obtained applying the BJ methodology. $(s)ar_i$ parameters are the best fit for the (seasonal) auto regressive process of order i. $(s)ma_j$, the best fit for the order j (seasonal) mobile average parameters.

Process	ar_1	ar_2	ar_3	ma_1	ma_2	ma_3	sar_1	sma_1	sma_2
Val	-0.054	0.740	0.058	-0.674	-0.915	0.599	-0.524	0.021	-0.077
Std Err	0.058	0.108	0.079	0.059	0.076	0.104	-	0.099	0.156

Figure 2.13b shows an instance of a generated price series (in black) using the model discussed in the previous section. The figure also shows simulations of predicted prices for various σ_{pred} values. As a measure of the accuracy of the simulated predictions, we use the mean squared error (MSE) defined as $MSE(P, \hat{P}) = \frac{1}{T}\sum_{t \in T}(P_t - \hat{P}_t)^2$. Values of MSE for those particular simulations are also reported on the figure. Small values (e.g. 4.46) of MSE suggest that the overall predictions are good. As the accuracy of predictions degrades, the value of MSE rises (e.g. 124.83). Throughout the remainder of this case-study we will be using the σ_{pred} parameter to simulate situations ranging from perfect ($\sigma_{pred} = 0$) to highly inaccurate price forecasts with bigger values of σ_{pred}.

2.4.2 Minimizing Data Centre Electricity Cost

This section presents an example for modeling real-world problem using the formalism of mathematical modeling. We formalise the problem of finding the minimum total electricity cost for a network of data centres. The formulation is adapted from the problem described in [RLXL10]. Our intention, however, is to give a systematic characterization of price properties on the cost of running a network of data centres. Table 2.2 summarizes the parameters and the decision variables needed to formulate the problem.

We first assume that each data centre is in a location where electricity price $P_i(t)$ on the wholesale market varies every 30 minutes. We consider a set L of locations spread geographically so that there is a time lag $TL \in \{0, 1, 2, \ldots, 24\}$ thirty minutes between two consecutive locations. This time lag parameter actually controls how the price signals at the various locations will be shifted with respect to each other; it has been suggested that energy prices become less correlated between two locations as the distance between them increases, i.e. the further away two locations are, the less correlated are their energy prices [QWB+09]. Using the model defined in previous section, we also define $\hat{P}_i(t)$ as a vector of predicted prices of $P_i(t)$.

Table 2.2: Notations for parameters and decision variables

N	Number of Locations
M_i	Number of servers available at $i \in L$
μ_i	Request rate handled by a server at $i \in L$
Po_i	Power used by a working server at $i \in L$
$WL(t)$	Amount of requests for period t
$P_i(t)$	Electricity Price at i during time slot t
$\hat{P}_i(t)$	Forecasted Price at i during time slot t
TR	Time needed to reconfigure
TL	Time lag between two consecutive locations
m_i	Number of turned on servers at $i \in L$
λ_i	Number of requests assigned to $i \in L$

Each data centre has a number M_i of servers that can be switched on or off in order to handle the workload WL at location i. At each period the decision is thus to turn on a subset $m_i \in \{0, \ldots, M_i\}$ of servers at each location i. We assume that each server in location i has a capacity factor μ_i, expressed in terms of processing requests, and will consume an amount Po_i of electricity if running. For each time interval t we can express the expected total energy cost of running N data centres as:

$$C_t = \sum_{i=1}^{N} m_i \times P_i(t) \times Po_i$$

Therefore, we can define the cost over all time periods as $C = \sum_{t \in T} C_t$. The decision of the assignment of workload at a particular time t is computed in order to minimize overall expected energy cost. The optimization process is thus based on the forecasted prices $\hat{P}_i(t)$. This quantity is given by:

$$\hat{C}_t = \sum_{i=1}^{N} m_i \times \hat{P}_i(t) \times Po_i.$$

In some of the experiments shown in the next section, we assume a perfect prediction accuracy ($\sigma_{pred} = 0$), and solving the problem with $\hat{P}$ is equivalent to solving the problem with P. A solution to this problem requires that all the workload is distributed among the locations. Thus we can express the workload constraint as $\sum_{i \in L} \lambda_i = WL(t)$. On the other hand, the assigned load to an data centre at location i should not exceed its processing power (requests): $\lambda_i \leq m_i \times \mu_i$.

Solving this problem requires finding the assignment of the workload to the data centres λ_i, and subsequently the number of servers, m_i, that are turned on. The mathemat-

ical model can be written as follows and will be solved for each time interval $t \in T$ considered in the problem:

$$\min_{\lambda_i, m_i} \sum_{i \in L} m_i \times \hat{P}_i(t) \times Po_i$$

Subject to

$$\lambda_i \leq m_i \times \mu_i, \qquad \forall i \in L$$

$$\sum_{i \in L} \lambda_i \leq WL(t),$$

$$m_i \in \{0, \ldots, M_i\}, \qquad \forall i \in L$$

$$\lambda_i \in \mathbb{N}. \qquad \forall i \in L$$

Finally, our model provides a way to simulate various levels of inertia in the system. The parameter TR specifies the number of time slots needed to reassign the workload. When TR is set to 0, we assume that the assignment for time t is performed instantaneously at the beginning of the period. For positive values of TR, a new assignment done at time t will be held over TR times slots before a new assignment is allowed to be performed. We simulate this by solving the optimization program on every time period divisible by TR and keeping the assignment in between those time periods.

2.4.3 Analysis

We consider the impact of factors such as price volatility, forecasting errors, time lag between locations, and time needed to reconfigure the system on the optimal energy cost required by a network of data centres. For doing so, all experiments were conducted with the same set of fixed parameters for both the set of locations and the workload. We simulated instances of the problem with 4 data centres such that the total maximum processing power was fixed at $\sum_{i \in L} \mu_i \times M_i = 167000$. On the other hand, the load was fixed at $WL_t = 100000, \forall t \in T$ requests. We thus have the guarantee

Table 2.3: Data centre setup for each of the four locations.

	data centres			
l	1	2	3	4
Po	100	110	120	110
μ	0.9	1.1	1.5	1.2
M	50000	40000	20000	40000

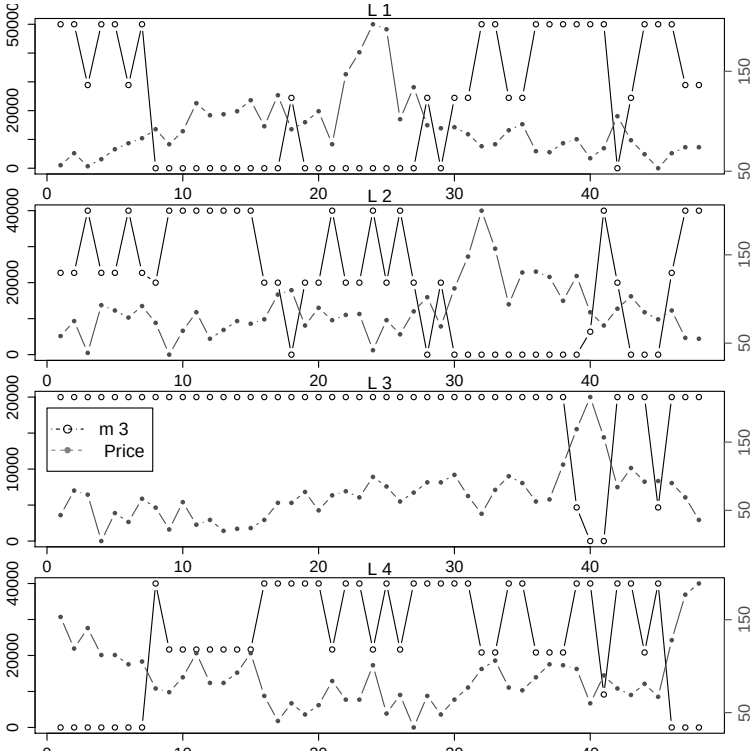
that the problem always admits a feasible solution.

The individual data centre configurations are summarized in Table 2.3. Those features form the static part of the model. We can see that data centre 3 has the most efficient configuration with a cheaper cost per unit of computation ratio. Despite the fact that we can order data centres by efficiency, finding an optimal assignment for a particular time period requires one to further investigate price behavior features. We thus discuss these parameters in the remainder of this section.

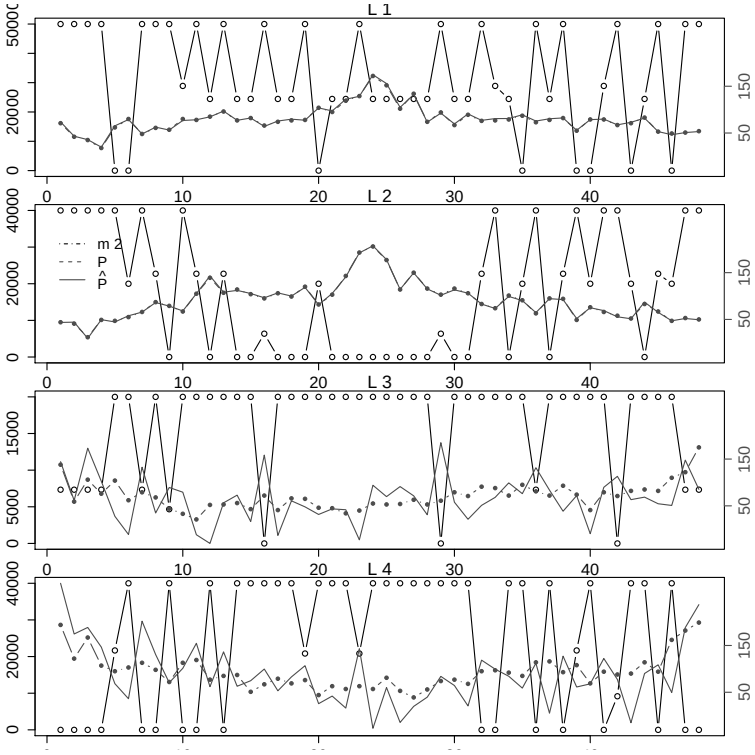
Figures 2.14 illustrates with two scenarios how the overall system behaves over time. Figure 2.14a shows how many servers (m_1 on top to m_4 at the bottom) were set to run in each of the 4 data centres over the 48 time intervals. In this particular scenario we defined a 4 hour (8 intervals of 30 minutes) timezone difference between each location causing price peaks to be shifted across the day. The workload distribution was computed assuming that price forecasting was perfect (i.e. $\sigma_{pred} = 0$). The price levels for each individual location are also reported on the right axis. Despite the fact that some IDs are more efficient than others, we see that none of the data centres are constantly working at full capacity. In fact, none of the four data centres are producing any work while local electricity prices are at their highest. Due to its superior configuration, data centre 3 is running all its servers over most of the day but is still powering down during time slots 39, 40 and 41, where local energy costs are highest.

We further note that the assignment over time is very sensitive to price variations. This is due to the fact that the cost per unit of computation in the various locations are always relatively close to each other. This ratio favors, in turn, different locations only because of the price differentials occurring within a day.

Let us explore another scenario involving errors in price prediction for both data centre 3 and data centre 4. For Figure 2.14b errors were simulated with a $\sigma_{pred} = 10$ level such that $\hat{P}_t = \mathcal{N}(P_t, 10), \forall t \in T$ and are represented with the solid gray line. In this scenario we observe that forecasted prices can significantly depart from the actual price. Since workload distribution is computed from the predicted prices $\hat{P}_t$, we can clearly see that this assignment is not optimal. For instance, the workload assignment at $t = 29$ seem to be erroneous as data centre 1 and data centre 2 are both in a quite high-priced period but are still carrying workload. This is due to a large overshooting of the price forecast in location 3 at that particular time causing data centre 3 to turn all its servers down and thus shifting the load to other data centres. Looking more closely at the dynamics of this scenario, one can spot this faulty behavior taking place over the 24 hour period, e.g. $t = 16$, $t = 24$.



(a) 4 hour timezone gap between each consecutive locations.



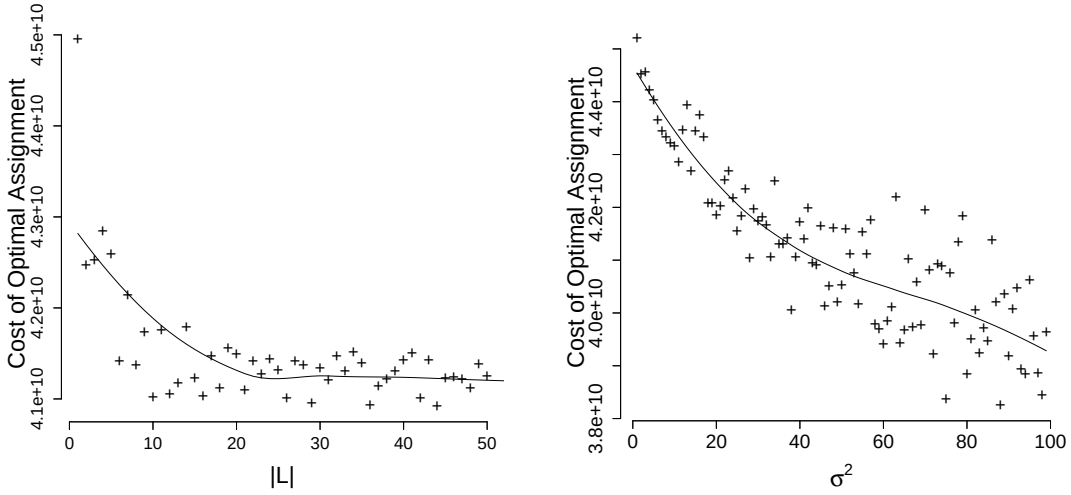
(b) Inaccurate price forecasts for locations 3 and 4.

Figure 2.14: Number of servers running m_i (in black) and price levels (in gray) at each locations. In Figure 2.14b we also plot the electricity price forecasts (solid gray line). Since σ_{pred} was set to 0 for data centres 1 and 2, actual and forecasted prices are strictly overlapping at these locations.

We now report on systematic experiments that give insights into how various features of the problem can affect the overall energy cost. Unless specified, the configuration mentioned above was used. Each experiment was run 10 times. In the following figures, we systematically fit curves to the data to help demonstrate the trend in the results.

Price Variability. The first two experiments that were conducted aim to characterize the impact of variety among prices on the overall assignment cost. To this end we increase, in turn, the number of locations considered in the problem. The underlying assumption here is that the number of opportunities to reduce electricity costs occur more frequently as the number of locations increases. Price P_i were all generated with $\sigma^2_{price} = 126.6$ and their forecast were set to be perfect (i.e. $\sigma_{pred} = 0$). Neither time lags nor reconfiguration times were used. One should note that the capacity M_i of each data centre was tuned in order to keep the total processing power constant (i.e. 167000).

As Figure 2.15a shows, the cost of the optimal assignment quickly decreases with the cardinality of L. In fact, it dropped by almost 10% from a situation in which there is no possibility to take advantage of price differentials ($card(L) = 1$) to a situation in which price differentials are induced by a larger number of locations ($card(L) = 20$ and above). This effect seems to level off for more than 20 locations in this particular



(a) The effect on optimal energy cost associated with increasing the number of locations.

(b) The effect associated with intrinsic price fluctuation with $\sigma^2_{price} \in \{0, 5, \ldots, 500\}$.

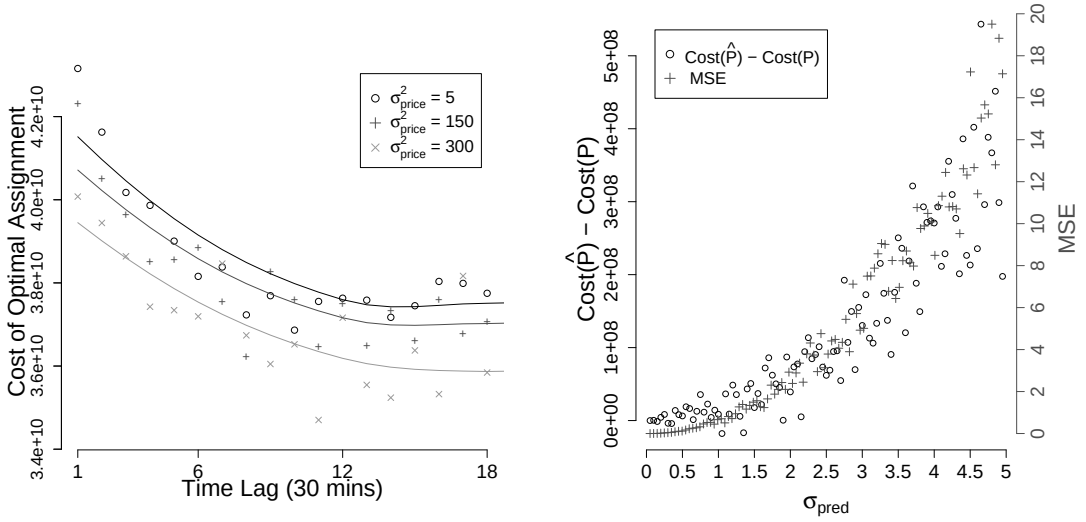
Figure 2.15: Exploiting price differentials to reduce overall operating costs. Both factors can be seen as opportunities to exploit price differentials in order to reduce overall operating costs.

setup.

To further test the impact of price variety on total energy cost, we can also generate prices that are intrinsically more or less fluctuating as shown in Figure 2.13a by varying the σ^2_{price} parameter. Figure 2.15b shows how the level of fluctuation among prices affects the cost of the optimal assignment. For low values of σ^2_{price}, prices at distant locations will not deviate much from each other. As σ^2_{price} rises, prices are more and more noisy, and thus exhibit more intrinsic diversity. We see the impact of that diversity by the decreasing cost of the optimal assignment. In the best cases, it appears that energy costs can be reduced by almost 15% if prices at the different locations show a reasonable level of variability. Finally, we note that it is not clear if this effect would level, but we clearly see that as σ^2_{price} progresses the cost displays more variance.

Timezone Effect.

The timezone effect was illustrated in the first scenario (Figure 2.14a) discussed in the previous section. We can show that, given the particular daily shape of real-time electricity prices, spreading data centres over distant locations gives substantial electricity cost savings. To demonstrate thus, we assumed perfect prediction on prices generated with a $\sigma^2_{price} \in \{5, 150, 300\}$ and varied the TL parameter to set the time lag between each consecutive locations. Thus, $TL = 0$ means that prices are perfectly in phase and $TL = 2$ means that each consecutive location is separated by an hour, slightly shifting the price signals.



(a) Varying the time lag between locations.

(b) Varying the quality of price predictions.

Figure 2.16: Average optimal assignment cost under several time lags and price prediction regimes.

As Figure 2.16a suggests, in this configuration a time lag of 12 30-minute intervals (6 hours) gives the best results. This is not surprising since with $TL = 12$ prices at the four locations are perfectly out of phase. This gives the opportunity to route the load away from locations showing high price levels (midday) to locations where electricity is cheaper (night time). In this particular setup, a perfect geographical spread could account for up to 15% in electricity cost savings. We further note that the price variability effect does not contradict the time lag effect since the observed trends are quite similar.

Price Forecast Quality. As the "bad forecast" scenario depicted in Figure 2.14b suggested, low prediction accuracy can lead to non-optimal assignments. To gain an insight into how the quality of price forecasting affects the cost of assignment, we varied the parameter σ_{pred} in the range $0, \ldots, 5$ by steps of 0.1. Predictions will thus be fuzzier as σ_{pred} rises. Prices at the various locations were generated with a standard variability level $\sigma^2_{price} = 126.6$. Recall that when $\sigma_{pred} = 0$, predictions are perfect and thus the assignment will be optimal.

Figure 2.16b shows that the difference between the assignment cost computed with $\hat{P}$ and the same solution evaluated with the actual price P. This difference can be interpreted as a penalty cost induced by bad decision-making due to the uncertainty while predicting prices. We show that this penalty cost is rising with the level of uncertainty σ_{pred}. Furthermore, this difference seems to be strongly correlated with the MSE indicator measuring the accuracy of predictions.

Reconfiguration Time. The last feature that was tested is the speed with which the data centre can be reconfigured. Until now, we assumed that the system could be configured instantaneously at the beginning of a given time period. For realism sake, we defined scenarios in which the time needed to reconfigure the system was set to $TR \in \{0, 1, 2, 4\}$. For instance, when TR is set to 2 the system will need an hour to reassign the workload.

As can been in Figure 2.17, the reconfiguration time dramatically affects the cost of the assignment. For $TR = 0$ we assumed that reconfiguration for period t is done at the beginning of period t, thus we have the same behavior as shown in Figure 2.15a. For $TR = 4$, we see that the gain induced by more variability within prices is almost null. In fact, several runs are indicating a degradation of the cost of assignment. This could be explained by the fact that, introducing latency in the system, prevents one to take immediate advantage of price differentials. We can derive from this that the more that prices fluctuate, the more flexible the system must be in order to benefit from it.

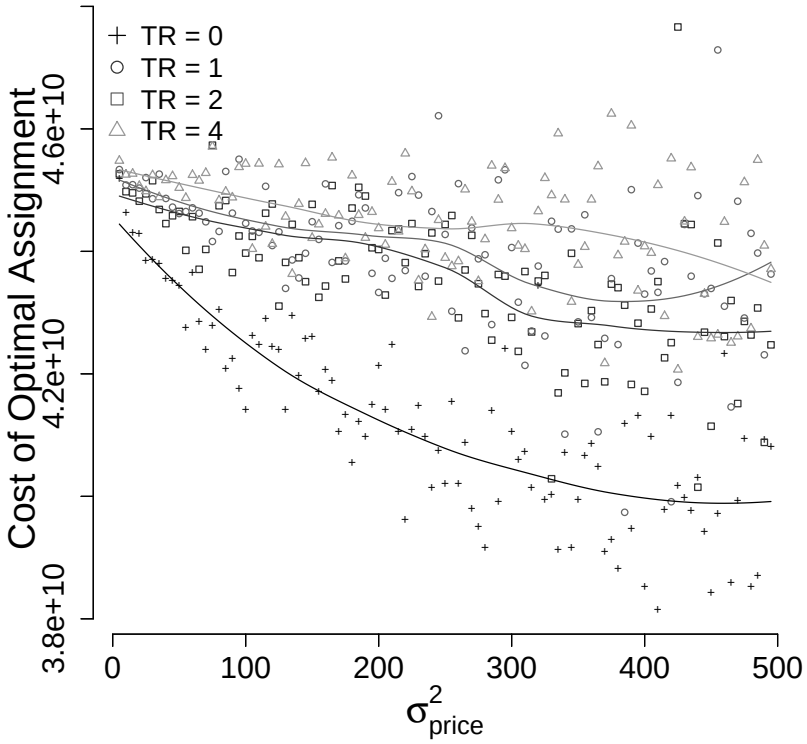


Figure 2.17: Time to reconfigure $TR \in \{0, 1, 2, 4\}$. As there is more inertia in the system, the expected gain from highly fluctuating prices vanishes.

2.5 Conclusion

In this chapter we have introduced concepts supporting subsequent technical chapters. We discussed techniques to model and solve combinatorial problems. The efforts carried by the research community to optimise often inefficient cloud infrastructures was then reviewed with a particular focus on workload consolidation. Finally, we showed through an example tackling a setting in which data centres are spread over several distant locations how to use statistical modeling and optimisation techniques. The basis of our analysis is an approach to forecasting on the basis of a time-series representing energy prices over time, and an approach to controlling the effect of forecast errors. An assumption has been that the more distant a pair of data centres, the less correlated are their local energy prices.

Chapter 3

A Generalisation of Bin Packing as a Core Consolidation Problem

Summary. *This chapter formalises a packing problem that emerges as a core sub-problem for managing workload consolidation in data centres. As a generalisation of the bin packing (*BP*) problem, it considers a set of tasks (items) to be assigned to a set of machines (bins) under capacity constraints on each machine. Unlike classic* BP *settings, items have a lifespan in the bins. We define the cost of using a bin as the product of the bin's capacity and the time it will be used for. We refer to this problem as the Temporal bin packing problem (*TBP*).*

*We formalise the problem using mathematical modeling and present optimisation models using Mixed Integer Programming (*MIP*) and Constraint Programming (*CP*) for two contrasting but equivalent viewpoints on the problem. The packing model (*PA*) extends traditional* BP *models while the temporal model (*TP*) explicitly models time with a sequence of packing problems. In addition, two ad-hoc symmetry breaking techniques are developed. Finally, we introduce both a lower bound and an upper bound on the objective function.*

Our empirical results suggest that the TBP *is a challenging problem for complete solvers to prove optimality. While breaking symmetry considerably reduces the computational effort for both* PA *and* TP *models, the packing model using* CP *should be considered for solving larger instances of the* TBP.

As a first step to model consolidation problems in cloud systems we use the formalism of the previously discussed bin packing problem (BP). Informally, we are given a set of machines (bins) and a set of tasks (items). Each task is associated with a resource requirement (i.e. CPU cycles) and the duration for which it will exist in the system. The goal is to assign all the tasks to machines while minimising the overall allocated resources. This allows us to switch off those machines that are spared by achieving better workload consolidation across the pool of machines. Motivated by this, we define the cost of using a machine as the product of its resource capacity and the time it will be used for. We refer to this problem as the Temporal bin packing problem (TBP). Although, TBP has strong connections with BP, the cost of using a bin captures the notion of time to live the bin.

As highlighted in Chapter 2, cloud providers often build and maintain over-provisioned infrastructures. As such, it is reasonable to consider that at any point in time, there are enough available machines for the workload to be addressed without any delay due to the lack of available resources. A desirable aspect of workload allocation policies is to implement workload consolidation in order to spare machines not needed to tackle the workload [BB10a]. This is usually achieved by minimising the allocated resources across the data centre. In the context of our application, minimising the sum of the machines usage cost as defined later is equivalent to minimising allocated resources across the infrastructure. In our model, we thus aim to minimise the allocated resources under the assumption that each task will be assigned and processed as soon as it arrives. Our model focuses on an offline formulation of this variant of the bin packing problem.

The remainder of the chapter is structured as follows: We provide the motivation for defining TBP in the context of workload consolidation for data centres and develop a mathematical model in Section 3.1. Section 3.2 describes how the time dimension can be either implicitly or explicitly modelled with both MIP and CP. In order to strengthen the models, Section 3.3 introduces symmetry breaking techniques. In Section 3.4, both a lower and upper bound on the objective function are derived. Finally, Section 3.5 compares the performance of the various models introduced in the chapter.

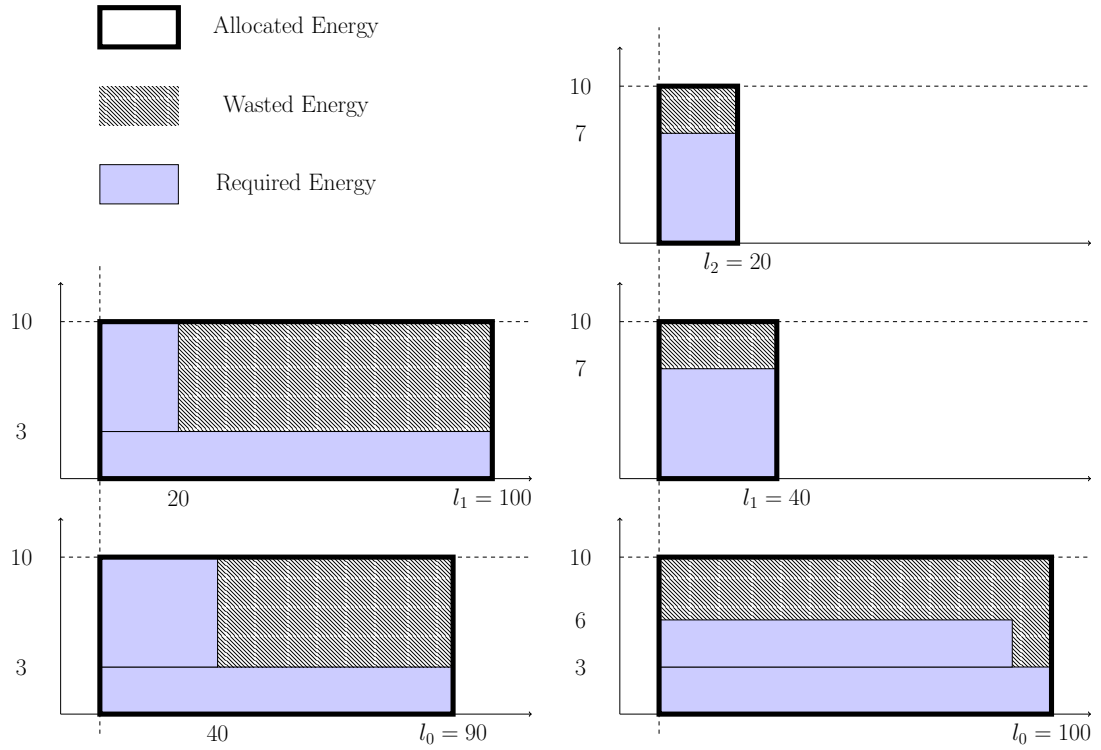
3.1 The Temporal bin packing Problem

Assigning items to bins while respecting capacity constraints is a core problem in workload consolidation problems faced by cloud providers. Servers in data centres are typically constrained over several dimensions (i.e. resources such as CPU time,

RAM and I/O capacity or network communication [GHMP08]). We will focus on a single dimension as CPU cycles are typically reported to be a bottleneck in such systems [RTG+12]. For all the tasks submitted to the system, it is desirable to minimise the gap from submission time to the time the task starts running on the system. Under the assumption that the infrastructure is over-provisioned, we consider this gap to be negligible as it is always possible to allocate more machines if needed. Also, we make the assumption that once a machine is powered-up the number of VMs assigned to it does not make a difference, only the maximum duration of items assigned to it does. Although workload consolidation problems are usually seen as online problems, we model an offline problem that should be solved sequentially in order to provide solutions to the online problem. We provide a formalisation of this core offline problem in which the allocation of tasks to servers may be seen as a generalisation of the bin packing problem. To the best of our knowledge this problem has not been described in the literature. For related packing problems, refer to Section 2.3.4.

Let $\mathcal{J} = \{1, \ldots, n\}$ be the set of items (tasks to be assigned) and each item $i \in \mathcal{J}$ be defined by a pair (q_i, r_i) where $q_i \in \{1, \ldots, Q\}$ denotes its size and $r_i \in \mathbb{N}^+$ denotes the duration for which it will be assigned to a bin. Let $\mathcal{M} = \{1, \ldots, m\}$ be the set of bins. A bin $j \in \mathcal{M}$ is defined by a capacity $C_j \in \{1, \ldots, Q\}$. A bin is said to be *opened* if it contains at least one item. The cost of using a particular bin j is a linear function $C_j \times l_j$, where l_j is the maximum duration of all the items assigned to it. The problem is to find an assignment from items to bins subject to capacity constraints while minimising the sum of the bins usage costs $\sum_{j \in \mathcal{M}} C_j \times l_j$. We refer to this problem as the Temporal bin packing problem (TBP). Throughout the rest of the chapter we will fix the bins capacity to the constant Q such that $\forall j \in \mathcal{M} : C_j = Q$ with Q being the size of the overall largest item.

In such a context, we note that minimising resource wastage and minimising allocated resources are equivalent problems if all bins have the same capacities Q. We define the energy as duration times requirement Let the required energy be the sum over all items of the product of each item resource requirement and its duration, i.e. $R = \sum_{i \in \mathcal{J}} q_i \times r_i$. This energy is constant for any instance of TBP. Indeed, R does not depend on the assignment of items to bins. Let the allocated energy depend on the items assignments and be expressed as $A = \sum_{j \in \mathcal{M}} Q \times l_j$. Finally, the wasted energy is the difference between the allocated energy and the required energy $W = A - R$. Minimising wasted energy is: $\min \sum_{j \in \mathcal{M}} (l_j * Q) - \sum_{i \in \mathcal{J}} (r_i \times d_i) \equiv \min \sum_{j \in \mathcal{M}} (l_j * Q)$. Since R is constant and we consider that all bins have the same capacity, minimising the wastage energy is equivalent to minimising the allocated energy. The required energy, allocated energy and wasted energy are illustrated on Figure 3.1. The figure shows two contrasting



(a) An optimal BP solution which uses only 2 bins. Non optimal for TBP. Its cost is 1900.

(b) A non optimal BP solution using 3 bins instead of 2. An optimal solution for TBP with a cost of 1600.

Figure 3.1: An optimal solution for BP does not necessarily yield a good solution for TBP.

solutions to the assignment of 4 items.

Furthermore, we provide the intuition TBP is a generalisation of BP where $\forall(i, i') \subseteq \mathcal{J} \times \mathcal{J} : r_i = r_{i'} = r$. In that case the objective function reduces to $\sum_{j \in \mathcal{M}} Q \times l_j$ with the domain of l_j is reduced to $D(l_j) = \{0, r\}$. The items duration being all equal, we can guarantee that $l_j = r$ for all *opened* bins reducing the problem to a simple packing problem over one dimension.

Although TBP is closely related to BP, an optimal solution for BP does not necessarily yield a good solution for TBP. Consider an empty state of the bins with capacity $Q = 10$. Let the following four items form an instance of TBP: $J = \{(3, 100), (3, 90), (7, 40), (7, 20)\}$ where each tuple stands for size and duration (i.e. (q_i, r_i)). As seen in Figure 3.1a, finding a feasible assignment for the four items can be done using a BP approach. This solution uses the minimum number of 2 bins. Nevertheless, when the notion of time (r_i) is introduced in the problem, the objective function we are interested in is evaluated as the sum of the allocated energy (i.e. $10 \times 100 + 10 \times 90 = 1900$). In contrast, as shown on Figure 3.1b, an optimal TBP solution uses 3 bins with a cost

evaluated at $10 \times 100 + 10 \times 40 + 10 \times 20 = 1600$, which is the best assignment of these four items. With the same reasoning, using arguments exposed previously, it is easy to understand that this solution also minimises resource wastage in the bins.

3.2 Packing Versus Temporal Models

We provide two contrasting but equivalent optimisation models for the TBP. First, the *Packing* model (PA) focuses on finding an assignment in such a way that the capacity constraints on the bins are satisfied. While this model is similar in nature to the classic formulation of BP, it defines l_j variables capturing the time for which each bin j will be allocated.

In contrast, the *Temporal* (TP) model explicitly models time by posting a series of packing constraints ensuring that the capacity constraints are satisfied at each time point. Although equivalent to PA, this model is more explicit and thus grows faster with instance sizes. In the next section, we introduce a Mixed Integer Programming (MIP) and a Constraint Programming (CP) implementations for both the PA and TP models.

3.2.1 Packing Model (PA)

MIP implementation. We can formulate the TBP as a MIP model using Boolean variables to decide on the assignments, i.e. $x_{ij} = 1$ if item $i \in \mathcal{J}$ is assigned to bin $j \in \mathcal{M}$, 0 otherwise. We introduce two other sets of variables. The first set models the time for which each bin will be allocated. Let $\forall j \in \mathcal{M} \ : \ l_j \in \mathbb{N}$. Note that a tighter upper bound on l_j variables is the maximum duration of all items considered in the instance i.e. $\max_{i \in \mathcal{J}}\{r_i\}$. The argument is that the duration of the longest runing task is a natural upper bound on the run time of the machines. The second set exposes the usage levels u_j of the bins. Let $\forall j \in \mathcal{M} \ : \ u_j \in \{0, \ldots, Q\}$. With these variables defined the MIP model for PA reads:

$$\min \sum_{j \in \mathcal{M}} l_j \cdot Q \tag{3.1}$$

$$s.t. \sum_{j \in \mathcal{M}} x_{ij} = 1 \qquad \forall i \in \mathcal{J} \tag{3.2}$$

$$u_j = \sum_{i \in \mathcal{J}} x_{ij} \cdot q_i \qquad \forall j \in \mathcal{M} \tag{3.3}$$

$$u_j \leq Q \cdot l_j \qquad \forall j \in \mathcal{M} \tag{3.4}$$

$$l_j \geq x_{ij} \cdot r_i \qquad \forall i \in \mathcal{J} \quad \forall j \in \mathcal{M} \tag{3.5}$$

$$\sum_{j \in \mathcal{M}} l_j \cdot Q \geq \sum_{i \in \mathcal{J}} r_i \cdot q_i \tag{3.6}$$

$$\sum_{j \in \mathcal{M}} u_j = \sum_{i \in \mathcal{J}} q_i \tag{3.7}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in \mathcal{J} \quad \forall j \in \mathcal{M} \tag{3.8}$$

$$u_j \in \{0, \ldots, Q\} \qquad \forall j \in \mathcal{M} \tag{3.9}$$

$$l_j \in \{0, \ldots, \max_{i \in \mathcal{J}}\{r_i\}\} \qquad \forall j \in \mathcal{M} \tag{3.10}$$

The program minimises the sum of the allocated energy over the bins (3.1) while stating that all items must be assigned to a bin (3.2). A bin that is hosting at least one item should have a positive allocation time which is enforced through constraint (3.5) by forcing l_j variables to larger values than the duration of any item that may be assigned to it. The assignment is subject to capacity constraints on every bin (3.3) with the usage defined as the sum of the item sizes assigned to it. Constraint (3.6) redundantly states that the overall allocated energy on the machines should be greater than the overall required energy. In the same way, constraint (3.7) redundantly states that the sum of the bin usages should match the sum of items requirements. These redundant constraints are featured in this model, as well as the following ones in order for the solvers to possibly cut parts of the search space. Finally, with (3.8) (3.9) (3.10) variables are taking values in their respective domains.

CP implementation. Let $\forall i \in \mathcal{J} \ : \ x_i \in \mathcal{M}$ be a variable that denotes the bin assigned to item i. l_j variables have the same semantics as above but their domain can reduced to the all distinct item duration : $D(l_j) = \{r_i \mid i \in \mathcal{J}\}$. We minimise a $cost$ variable (3.11) modeling the objective function defined as the sum over the bins allocation times l_j times their capacities Q (3.12). The right hand side of (3.13) stands as a lower bound on the cost since the minimum cost would be reached if all the items are fitted with the required energy being equal to the allocated energy. The latter is similar to constraint (3.6) from the MIP model. We use constraint (3.14) to enforce that

each item i be assigned to bin j such that the sum of the weights of each item, q_i, in each bin j is equal to the usage u_j. Similar to constraint (3.5), constraint (3.15) sets the allocation time of each bins to match the maximum duration of the items assigned to it. The model CP model for PA reads:

$$\min cost \tag{3.11}$$
$$sum(cost, \langle l_1, \ldots, l_m \rangle, \langle Q, \ldots, Q \rangle) \tag{3.12}$$
$$cost \geq \sum_{i \in \mathcal{J}} q_i \cdot r_i \tag{3.13}$$
$$bin_packing_load(\langle u_1, \ldots, u_m \rangle, \langle x_1, \ldots, x_n \rangle, \langle q_1, \ldots, q_n \rangle) \tag{3.14}$$
$$\forall j \in \mathcal{M} : l_j = \max_{i \in \mathcal{J}}((x_i = j) \cdot r_i) \tag{3.15}$$
$$\forall j \in \mathcal{M} : u_j \leq Q \cdot l_j \tag{3.16}$$

In this formulation, the $bin_packing_load()$ global constraint is an implementation of a packing constraint suggested by Paul Shaw [Sha04]. This particular implementation exposes the *load* variables. This constraint requires that each item i with weight q_i, be put into bin j such that the sum of the weights of the items in each bin is equal to the bin's load u_j with domains $D(u_j) = \{0, \ldots, Q\}$. This global constraint is implemented in Gecode [STL10] and accessed through the Minizinc constraint modeling language [NSB+07b].

3.2.2 Temporal Model (TP)

In contrast to the *packing* model developed in the previous section, the *temporal* model explicitly models time points for which the problem admits a departing item. Let $\mathcal{T} = \langle t_1, \ldots, t_o \rangle$ be the ordered set of these time points such that t_1 maps to the time point of value 0, t_2 the smallest duration, and t_o the longest duration of any item. Let $K^+ = \{1, \ldots, o\}$ be the index over $\mathcal{T}$ and $K = \{1, \ldots, o-1\}$ an index over $\mathcal{T}$ without its last element. Last, K^- is defined as K minus its last element. With time defined explicitly, we can build subsets of items $\forall k \in K \ : \ \mathcal{J}_k = \{i \mid i \in \mathcal{J} \wedge r_i > t_k\}$. Naturally, $\mathcal{J}_1$ includes all the tasks whereas $\mathcal{J}_{o-1}$ includes the subset of tasks with maximum duration.

MIP implementation.

Similar to the *packing* model, let $\forall (i,j) \in \mathcal{J} \times \mathcal{M} \ : \ x_{ij}$ be Boolean variables

deciding on the assignment of item i. In addition, let $\forall(j,k) \in \mathcal{M} \times K \ : \ u_{jk} \in \{0,\ldots,Q\}$ be integer variables modeling the utilisation level of machine j at time point k. Furthermore, we introduce a set of Boolean variables $\forall(j,k) \in \mathcal{M} \times K \ : \ y_{jk}$ stating whether or not machine j is processing any task at time k. The model is as follows.

$$\min \sum_{k \in K} \sum_{j \in \mathcal{M}} Q * (t_{k+1} - t_k) * y_{jk} \tag{3.17}$$

$$\sum_{j \in \mathcal{M}} x_{ij} = 1 \qquad \forall i \in \mathcal{J} \tag{3.18}$$

$$u_{jk} = \sum_{i \in \mathcal{J}_k} x_{ij} \cdot q_i \qquad \forall j \in \mathcal{M} \quad \forall k \in K \tag{3.19}$$

$$u_{jk} \leq Q \cdot y_{jk} \qquad \forall j \in \mathcal{M} \quad \forall k \in K \tag{3.20}$$

$$y_{jk} \geq y_{jk+1} \qquad \forall j \in \mathcal{M} \quad \forall k \in K^- \tag{3.21}$$

$$u_{jk} \geq u_{jk+1} \qquad \forall j \in \mathcal{M} \quad \forall k \in K^- \tag{3.22}$$

$$\sum_{j \in \mathcal{M}} u_{jk} = \sum_{i \in \mathcal{J}_k} q_i \qquad \forall k \in K \tag{3.23}$$

$$x_{ij} \in \{0,1\} \qquad \forall j \in \mathcal{M} \quad \forall i \in \mathcal{J} \tag{3.24}$$

$$u_{jk} \in \{0,\ldots,Q\} \qquad \forall j \in \mathcal{M} \quad \forall k \in K \tag{3.25}$$

$$y_{jk} \in \{0,1\} \qquad \forall j \in \mathcal{M} \quad \forall k \in K \tag{3.26}$$

The objective function (3.17), equivalent to that of the *packing* model (3.1) and (3.11), is computed piece-wise over all the machines and all consecutive time points. Items must be assigned to bins (3.18) under capacity constraints enforced by (3.19) and (3.20). Constraint (3.21) redundantly states that if a bin is allocated at time point k, it is implied that it should be running at previous time points as well. The same reasoning applies to the usage of the bins over time (3.22).

Finally, (3.23) states that the overall usage at any time period should match the sum of sizes of those items still allocated ($\mathcal{J}_k$). Constraints (3.24), (3.25), (3.26) are forcing variables to take values in their respective domains.

CP implementation. We provide a Constraint Programming formulation based on a conjunction of $bin_packing_load()$ constraints. Each constraint models a time point.

Let $x_i \in \mathcal{M}$ be an integer variable that denotes the bin assigned to item $i \in \mathcal{J}$. Let y_{jk} be a Boolean variable that denotes whether the bin j is used at time-point k. Let u_{jk}

be a integer variable denoting the level of utilisation of machine j at time-point k. The domain of u_{jk} is $D(u_{jk}) = \{0, \ldots, Q\}$. The CP formulation for the *temporal* model reads as:

$$\min \sum_{k \in K} \sum_{j \in \mathcal{M}} Q \cdot (t_{k+1} - t_k) \cdot y_{jk} \tag{3.27}$$

$$\forall k \in K \ : \ bin_packing_load(\langle u_{1k}, \ldots, u_{m|J_k|} \rangle, \langle x_1, \ldots, x_{|J_k|} \rangle, \langle q_1, \ldots, q_{|J_k|} \rangle) \tag{3.28}$$

$$\forall j \in \mathcal{M} \quad \forall k \in K \ : \ y_{jk} \Leftrightarrow u_{jk} > 0 \tag{3.29}$$

$$\forall j \in \mathcal{M} \ : \ decreasing(\langle u_{j1}, \ldots, u_{j,|K|} \rangle) \tag{3.30}$$

$$\forall j \in \mathcal{M} \ : \ decreasing(\langle y_{j1}, \ldots, y_{j,|K|} \rangle) \tag{3.31}$$

$$\forall_{k \in K} \sum_{j \in \mathcal{M}} u_{jk} = \sum_{i \in \mathcal{J}_k} q_i \tag{3.32}$$

The objective function (3.27) to minimise here is the same as the one introduced in the previous MIP formulation. Constraint (3.28) imposes that each item $i \in J_k$ be put into bin x_i such that the sum of the weights of each item, q_i, in each bin j is equal to the usage u_{jk} at time-point k. In this model, the y_{jk} variables are constrained (3.29) to be 1 if and only if the usage of machine j at time point k if higher than 0. In (3.30) the usage of any machine is constrained to decrease over the time points while constraint (3.31) is stating that if bins j was opened in time k, it is necessary for the bins to be opened in time $k - 1$. Finally, (3.32) is the equivalent of (3.23) from the MIP model.

We have shown a MIP and a CP implementation for both PA and TP models for TBP. In the next section, we show how these models can be strengthened in order to improve pruning from the solvers.

3.3 Breaking Symmetry

We introduce some symmetry breaking rules that can be implemented in addition to the core models shown in previous sections. As can be seen on Figure 3.1b, the optimal solution for the instance illustrating the problem can be used to build equivalent solutions by permuting the bins. To reduce the search space, an order can be imposed over the bins. This order can be defined over the time for which bins are allocated (l_j

variables) or on the bins utilisation (u_j variables).

3.3.1 Breaking Symmetry on the PA model.

On Run Times (RT). We constrain the order of the bins to be decreasing on run time. For the PA-MIP model, this can be implemented by imposing the following set of constraints : $\forall j \in \{1, \ldots, m-1\} \ : \ l_j \geq l_{j+1}$. Similarly, the PA-CP should impose the values of l_j variables to be decreasing : $decreasing(\langle l_1, \ldots, l_m \rangle)$.

In addition, by enforcing this order, we can deduce that the running time of the first bin should be equal to the duration of the longest item $l_1 = \max_{i \in \mathcal{J}}(r_i)$ and that its usage has to be higher than the size of the biggest item $u_1 \geq q_i$ with i being the item with the longest duration. This holds true and can be implemented in both MIP and CP because the longest item will be hosted on this machine along with possibly other shorter items.

On Usage (US). An alternative way to break symmetry is to enforce the bins' order based on their usage level. This order can be imposed by posting $\forall j \in \{1, \ldots, m-1\} \ : \ u_j \geq u_{j+1}$ in the PA-MIP model. Similarly, the PA-CP should implement the constraint : $decreasing(\langle u_1, \ldots, u_m \rangle)$.

Additionally, breaking symmetry on u_j variables implies that the usage of the first bin is at least that of the size of the biggest item, i.e. $u_1 \geq q_i$ with i being the size of the largest item.

3.3.2 Breaking Symmetry on the TP model.

Similar techniques can be applied in the *temporal* model. Because the TP is more expressive, i.e. it models time explicitly, breaking symmetry can be done over the bins throughout the multiple time steps.

On Run Times (RT). In this model, with the MIP implementation, one can enforce the order on the run time by stating that a bin j can be allocated only if the bin $j-1$ is also allocated. This holds true for all time points considered in the problem. The ordering can be enforced through y_{jk} variables, i.e. $\forall j \in \{1, \ldots, m-1\} \quad \forall k \in K \ : \ y_{jk} \geq y_{j+1k}$. Likewise, in the CP implementation, the model should impose $\forall k \in K \ : \ decreasing(\langle y_{1k}, \ldots, y_{mk} \rangle)$.

With the ordering on run times, further properties can be deduced on the structure of solutions for TBP. Because the longest item will be assigned to the first machine, we can state that it should be allocated for all time points i.e. $\forall k \in K \ : \ y_{1k} = 1$.

On Usage (US). Finally, the order on the bins can be based on their utilisation level. This is only true for the first time point in the problem and should not be enforced for all time points. The MIP model should then be equipped with the following set of constraints : $\forall j \in \{1, \ldots, m-1\} \ : \ u_{j1} \geq u_{j+1,1}$. On the other hand, the CP model should implement $decreasing(\langle u_{11}, \ldots, u_{m1} \rangle)$.

3.4 Lower and Upper Bounds

Modern complete solvers rely on an implementation of a backtracking system. The backtracking system is responsible for systematically exploring the search space until having found an optimal solution. These systems are usually reasoning on both lower and upper bounds on the value of the objective function to guide the search procedure.

A solution yielding an upper bound on the objective function is a complete assignment from variables to values in their respective domains in such a way that all constraints are satisfied. Although such a feasible solution is in general not optimal, it can be used by solvers to prune sections of the search space that are yielding objective values that are dominated. A lower bound on the objective function is provided by a solution that is in general not feasible because some constraints of the original problem are relaxed.

We introduce with Algorithm 1 an ad-hoc method to compute a lower bound on the allocated energy (equivalently, on the resource wastage) on any instance of TBP. It relies on sorting the items by decreasing duration (line 2) and uses the $L1$ bound derived by Silvano Martello and Paolo Toth for the BP Problem [MT90a]. By definition, $L1$ is a lower bound on the number of bins used in a bin packing instance. The relaxation considers the items breakable and thus can be split and assigned to multiple bins at the same time. The $L1$ bound is provided on line 3 with $L1 = \lceil \sum_{i \in \mathcal{J}} q_i / Q \rceil$. We define on line 4 a vector R containing the cumulative sum of items sizes such that $R_1 = q_1, R_2 = R_1 + q_2, \ldots, R_n = R_{n-1} + q_n$. The main loop on line 5 iterates over the minimum number of bins needed to accommodate all items while finding for each of those bins their maximum running time l_m by retaining the maximum duration of any item assigned to it in this relaxed version of the problem (line 8 and 9).

To illustrate the algorithm, Figure 3.2 shows how it is applied to the small instance

Algorithm 1: $TBP_lowerBound()$

Input: $\mathcal{J}, \mathcal{M}, q, r, Q$
Output: lb

```
1  lb ← 0
2  J⃗ ← sortOnDurationDecreasing(J)
3  L1 ← ⌈∑_{i∈J} q_i/Q⌉
4  R ← cumSum(J⃗)
5  for m ∈ {0, ..., L1 − 1} do
6      l_m ← 0
7      for each i ∈ J⃗ do
8          if R_i > m * Q ∧ R_i ≤ (m + 1) * Q then
9              l_m ← max(l_m, r_i)
10     lb ← lb + l_m * Q
11 return lb
```

introduced in Section 3.2. The items are sorted by decreasing duration and assigned to $L1 = \lceil (3 + 3 + 7 + 7)/10 \rceil = 2$ bins considering that they are breakable. The l_m variables evaluate to 100 and 40 in this instance. Finally the lower bound is computed as $100 * 10 + 40 * 10 = 1400$.

As an upper bound on the objective function, we adapt the well known FIRSTFIT algorithm for BP. Algorithm 2 differs in the ordering of the items. We first order the items decreasingly on the items duration on line 1 and initialise both u_j and l_j variables on lines 3 and 4. Then, the loop over each each item searches for the first candidate bin

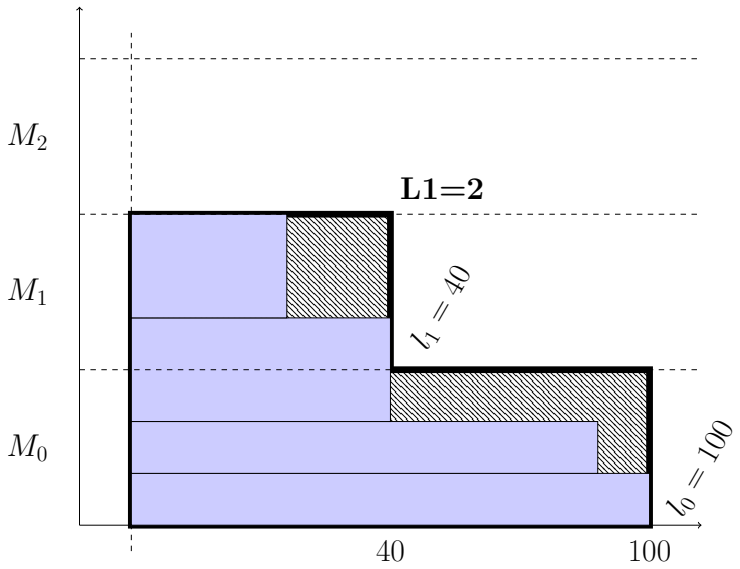


Figure 3.2: A relaxed version of TBP in which items are breakable. The lower bound evaluates to 1400.

to accommodate it. Naturally, line 7 checks whether or not there is sufficient remaining capacity in bin j for item i. If this is the case, we assign i to j and update the u_j and l_j variables accordingly on lines 8 and 9. The upper bound on the objective function is then returned on line 11 as the sum over the bins of their capacity Q times their run times l_m.

Algorithm 2: $TBP_firstFit()$

Input: $\mathcal{J}, \mathcal{M}, q, r, Q$
Output: ub

```
1  J⃗ ← sortOnDurationDecreasing(J)
2  for each j ∈ M do
3      u_j ← 0
4      l_j ← 0
5  for each i ∈ J do
6      for each j ∈ M do
7          if q_i + u_j ≤ Q then
8              u_j ← u_j + q_i
9              l_j ← max(l_j, r_i)
10             break
11 return ∑_{j∈M} Q × l_j
```

While *TBP_firstFit()* is a heuristic method, from the example introduced in Figure 3.1, it is clear that it would produce the optimal solution. Although it is the case in this instance, it does not hold true for any instance of the TBP. The lower bound and upper bound algorithms will be used to assess the solution quality returned by systematic solvers in the next section.

3.5 Empirical Analysis

As such the Temporal bin packing has not been described in the literature. In previous sections, we have presented alternative models (PA vs TP), implementations (MIP vs CP) and symmetry breaking techniques (RT vs US). We now consider the most efficient way to systematically solve the TBP. As it is not guaranteed to find optimal solutions as the instance size grow, we have thus provided both lower and upper bounds methods to compare the various approaches.

3.5.1 Experimental setup

As a framework for the experiments we implemented the MIP model using Numberjack [HOO10], a modeling package for constraint programming (version 1.1.0.) The backend solver used was CPLEX (version 12.51.). On the CP side, models were implemented with Minizinc [NSB+07b] (version 2.0.2) using as a back-end solver Gecode [Gec06] (version 4.4.0.) All experimental runs were performed on a cluster of Intel Xeon E5430 Processor with 12 GB of memory running CentOS release 6.6.

Due to the large spectrum of the experiments, we have unified the default search strategy in Minizinc to search on the assignment variables x_{ij} with a *first_fail* variable selection heuristic and *indomain_min* value selection policy. These strategies were experimentally found to be the best performing. Both Gecode and CPLEX were allowed to parallelize over 4 CPU cores. All optimisation models and instances used in the chapter are freely accessible. [1]

3.5.2 Instances

Two data sets were used to evaluate the performance of the various settings described in previous sections. The first one is a randomly generated set of instances ranging from size $n = 10$ to 100 items by increments of 10 giving rise to 10 instance classes : $C_{10}, C_{20}, \ldots, C_{90}, C_{100}$ each containing 10 different instances. The size of the bins and the size of the biggest possible item have been fixed to $Q = 100$. Each item has a uniformly generated size $q_i \in \{1, \ldots, Q\}$ as well as a uniformly generated duration $r_i \in \{10, \ldots, 1000\}$.

Our second data set is coming from a real-world application in the context of data centres. We extracted the data (item size and duration) from a trace recorded by Google in 2011 over a cluster of 12k servers [RWH11]. Much information has been gathered about the trace characteristics [RTG+12, DKF13, LC12]. Features of this data set were analysed in depth in Section 2.3.5 of this dissertation. For our evaluation, we have uniformly sampled this trace containing the description on several million tasks. As with the randomly generated instance, this data set contains 10 classes spanning from size 10 to size 100. Each class is a collection of 10 instances.

[1] `https://gitlab.insight-centre.org/mdecauwer/TemporalBinPacking`

Table 3.1: Percentage of instances solved within the time out function of the symmetry breaking technique

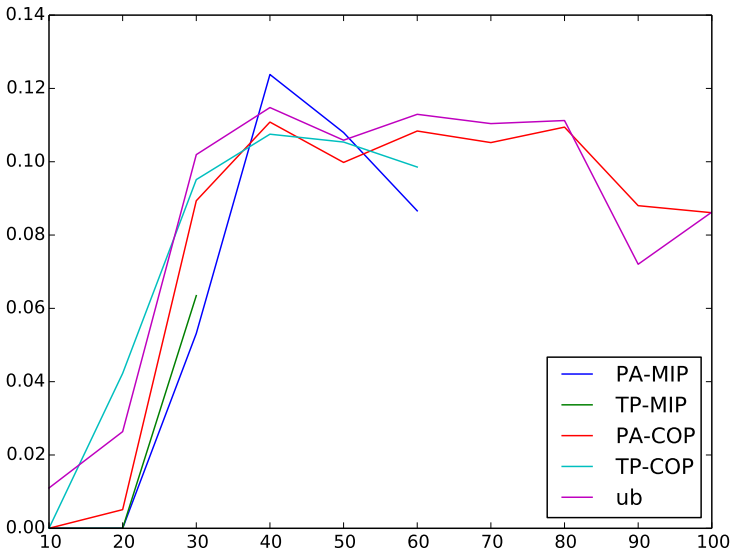
Time Out	Sym Break	Google	random
2sec	US	42.08	17.66
	RT	**45.65**	**28.20**
	NO	40.27	13.33
30sec	US	55.99	17.86
	RT	**69.52**	**41.43**
	NO	51.49	19.05
120sec	US	58.37	17.86
	RT	**68.16**	**47.72**
	NO	57.78	20.39
300sec	US	59.86	19.21
	RT	**70.05**	**53.04**
	NO	59.13	22.29

3.5.3 Analysis

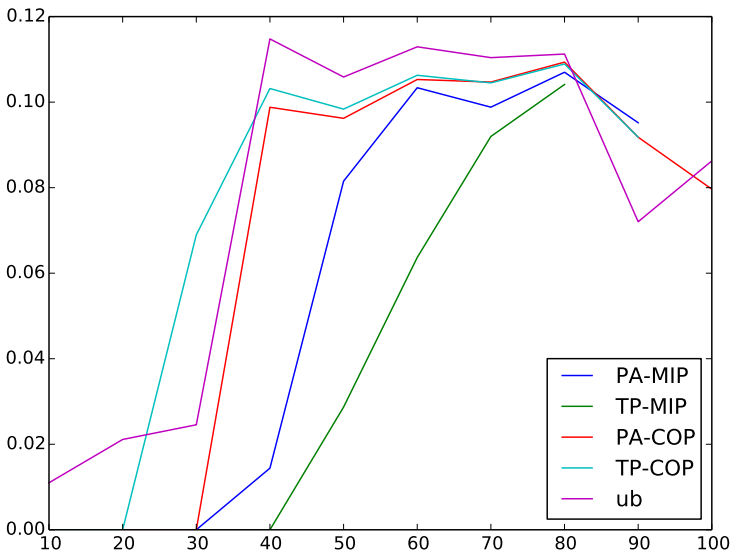
We investigate the performance of the various models for the TBP developed earlier. The two models (PA, TP), along with their respective MIP and CP implementations have been evaluated on both the uniform random data set and the instance set drawn from the Google trace. In order to simplify the analysis we are first interested in the performance of the symmetry breaking techniques introduced earlier. As suggested by Table 3.1 we see that breaking symmetry on the run times (RT) of the machines is the most efficient as it constantly outperforms breaking symmetry on the utilisation (US). We report in the table the proportion of instances that were solved to optimality within the timeout. NO means that no symmetry breaking technique was used. The proportion is computed over all combinations of models and solvers. The numbers show that symmetry breaking on the random instances has only a limited impact. There must be features in the Google data set, not present in the random data set, that take advantage of symmetry breaking.

From now on, only the RT strategy will be used to compare the various model / implementation couples. Due to the high rate at which data centres receive new tasks to allocate on their infrastructure, we will focus on the behavior of the models on a very short time out of 2 seconds. To get a better picture of the models' performances, we will also include a time out of 300 seconds.

Figure 3.3 reports on the average gap to the lower bound on the 10 classes of instances. This gap is computed as the average of distance between the objective reported by the solver and the lower bound given by Section 3.4 divided by the lower bound. Note that



(a) Time out 2 seconds.



(b) Time out 300 seconds.

Figure 3.3: Random instances. Average gap to lb after 2 and 300 seconds. The x-axis is instance size, the y-axis is the average gap to the lower bound computed across all instances of size x

if the solver proved optimality, then we use the optimal value as a lower bound.

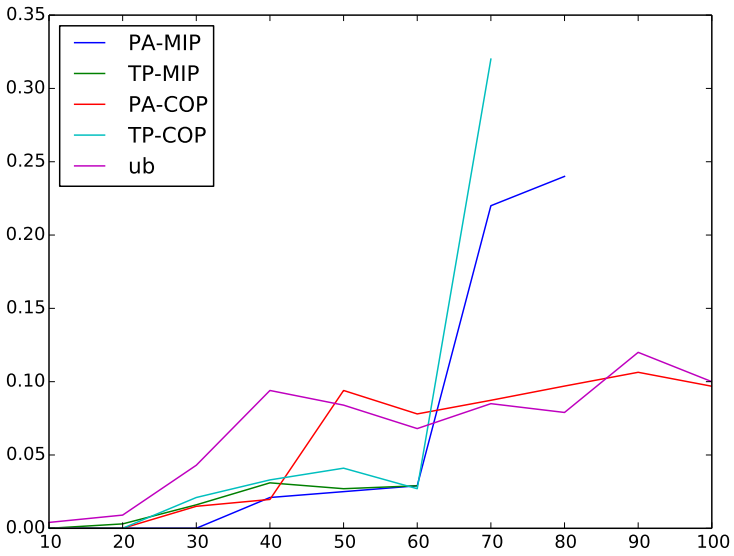
As the figure suggests, on short timeouts, of all the complete approaches, PA-CP is scaling better than other approaches. Overall its gap to the lower bound is leveling around 10% which seems comparable with the adapted $TBP_firstFit()$ heuristic. Under severe cut-off times, the TP model does not find solutions for instance sizes above 60 items. When allowed 300 seconds, surprisingly, most of the models are not finding solutions for instances of size 100. The only candidate that is scaling is PA-CP. The TP-MIP alternative seem to systematically yield better solutions than the other candidates models. Also, this model solves all instances up to size 40. When considering all models and solvers under both timeouts, the MIP family seems to hold better results than the models implemented with CP.

The performance of the models and solvers were evaluated on the instances drawn from the google data set and on presented in Figure 3.4. We can see on the figure that in general these instances are easier to solve. For a 300 seconds time out, all solvers were able to produce solutions up to instances of size 100. Under tighter timeout (2s), it is confirmed that PA-CP is the only setting able to produce solutions. This is most likely due to the fact that the model is of smaller size and that the default search strategy is able to get a first solution quickly. All the alternatives, provided 300 seconds of search time, are performing better than our heuristic upper bound.

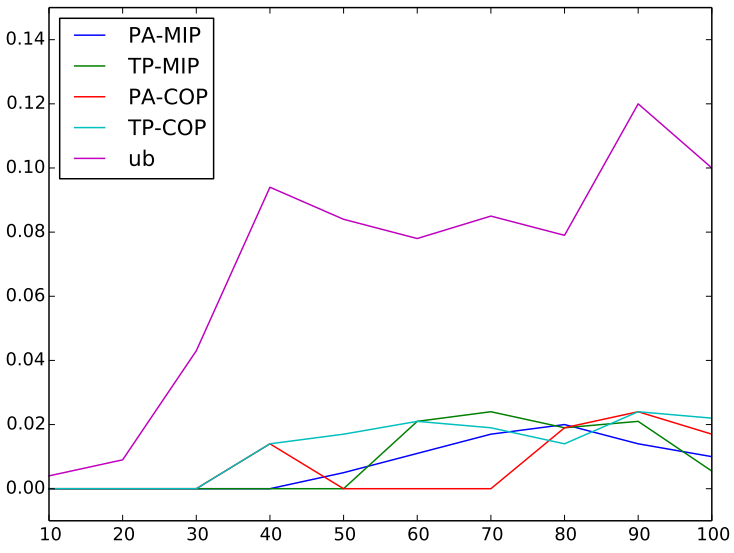
3.6 Conclusion and Limitations

Motivated by the problem of reducing resource wastage in data centres, we have developed a model that generalises the bin packing problem. The problem is to find an assignment of tasks to machines, subject to capacity constraints, such that the unused resources in time are minimised. The problem can be solved by implicitly considering the time dimension (*packing*), or by explicitly modeling time (*temporal* model). For both viewpoints on the problem, we have provided a MIP and a CP implementation along with symmetry breaking rules.

The evaluation of the PA and TP models on custom instances suggests that solving TBP is challenging even on rather small instances. Regardless of the model or solver, symmetry should be broken on the run times. Choosing which model to use depends on the size of the instance. We note that the straight- forward modification of well studied BP heuristics is yielding good performance. The best setting appears to be the BESTFIT on item size policy with a list of items ordered by decreasing duration.



(a) Time out 2 seconds.



(b) Time out 300 seconds.

Figure 3.4: Google Instances. Average gap to lb after 2sec versus 300sec. The x-axis is instance size, the y-axis is the average gap to the lower bound computed across all instances of size x

The TBP problem models a simplified workload management problem found in cloud computing infrastructures. The problem deals with simple uni-dimensional object with a known lifespan to be assigned to bins in an offline fashion. We have developed optimisation models capable of scaling up to 100 items. In most of the real operational settings in cloud systems, the nature of the optimisation problem is online as objects are revealed as time passes. The scale of the problem is also quite often orders-of-magnitude higher than the instances studied in this chapter. These aspects will be introduced and discussed in the next chapter in order to make this model for workload consolidation more realistic and compelling.

Chapter 4

Semi-online Consolidation with Uncertain Task Duration

Summary. *As highlighted in the introduction chapter, satisfying on-demand access to cloud computing infrastructures under quality-of-service constraints while minimising the wastage of resources is a key challenge in data centre resource management. This chapter deals with this challenge in a semi-online workload management system, allocating tasks with uncertain duration to physical servers. Our semi-online framework, based on a bin packing approach, allows us to gather information on incoming tasks during a short time window before deciding on their assignments. The contributions of this chapter are as follows: (i) we propose a formal framework capturing the semi-online consolidation problem; (ii) we propose a new dynamic and real-time allocation algorithm based on the incremental merging of bins; and (iii) an adaptation of standard bin packing heuristics with a neighborhood search algorithm for the semi-online context considered here. A systematic study of the impact of varying time-period size and varying the degrees of uncertainty on the duration of incoming tasks is then provided. The policies are compared in terms of solution quality and solving time on a data-set extracted from a real-world data centre trace.*

In this chapter, we leverage semi-online optimisation techniques in which workload allocations must be made without full knowledge of future demands. A semi-online formulation of the workload consolidation problem gathers information on incoming tasks for a short period of time. It may allow an operator to take more informed de-

cisions than the fully online formulation while keeping control of delays in task deployments. In a cloud computing production environment, it is often the case that the duration for which a task will consume resources is either approximated or not known at all. This fits the new challenge of on-demand allocations in which demands are guaranteed to be satisfied in real-time. We therefore formalise the workload consolidation problem as a semi-online packing problem whereby each bin maps to a machine and each item maps to a task.

We tackle the semi-online formulation of the on-demand workload consolidation where tasks have to be allocated to servers in real-time. While the vast majority of the work carried on workload consolidation considers either the offline [PB13, WTTL12] or online [PY10, HLW11, DC12] setting, we cast the problem in a semi-online framework by considering a short period of a few seconds within which tasks are grouped before being allocated to hosts. More precisely, we build upon the offline workload consolidation problem discussed in [CMO16].

The remainder of this chapter is organised as follows. Section 4.1 provides a mathematical formulation of the on-demand bin packing (ODBP) problem. In order to find solutions to the problem, Section 4.2 shows how to adapt packing heuristics to this consolidation problem. In Section 4.3, we contribute a flexible algorithm considering both tasks and machines to find efficient packings. Section 4.4 introduces neighborhood search in order to improve packings. Finally, Section 4.5 shows the performance of our approach compares to adapted heuristics of the related work in terms of solving time and solution quality. We demonstrate that our bin merging policy can achieve reductions in energy use of up to 40% over the compared approaches. We show that the policy is relatively robust to increased errors in the predicted duration of the tasks. Finally, we show that moving from the pure online problem to the semi-online problem, with relatively small decision time windows, has a significant impact on the solution quality, but that all policies quickly stabilise and do not benefit further from longer time windows.

4.1 Semi-Online Resource Wastage Minimisation

On-line policies decide the placement incoming tasks as soon as they arrive in the system. Such a framework must fully satisfy on-demand Quality-of-Service (QoS) requirements, guaranteeing the real-time placements of tasks. However, due to the lack of knowledge of which tasks might come next, on-line placement strategies may

provide poor consolidation solutions and waste more resources than required.

On the other hand, for efficient resource utilisation, off-line approaches consider the task placement as a batch optimisation problem for which the incoming tasks are known in advance. The knowledge of forthcoming tasks allows a better consolidation but involves more sophisticated techniques that may require an expensive solving time. In the context of on-demand placement, neither the existence of incoming tasks nor their duration can be known in advance.

The aim of our approach is to fill the gap between on-line and off-line approaches and investigate the benefit of a semi-online framework in terms of the trade-off between efficient resource utilisation and on-demand placement QoS. To have a better understanding of our overall objective in the semi-online context, consider an arbitrary start time of 0. For any time t, let z_m^t be the observed run-time duration for machine m between the time points 0 and t. Then, our overall objective is, for some sufficiently large time t, to allocate tasks to machines such that each task starts within δ seconds of its arrival time, and so that the sum over m of z_m^t is minimised.

Let us consider the following example of a sequence of six incoming tasks $a_0, \ldots, a_5$, having an expected duration of 50, 10, 100, 50, 10, 100 minutes respectively, requiring the same CPU resource equivalent to 50% of a machine capacity, and arriving at one second intervals. We aim to minimise the run-time of allocated machines. In the following, we consider semi-online consolidation as discussed in Section 2.2.3. Figure 4.1 shows an optimal placement of the incoming tasks within three different contexts, on-line, semi-online, and off-line. In an on-line context, the time for deciding the placement of the incoming tasks is negligible. The optimal placement, seen on the first row of Figure4.1, allocates the tasks a_0 and a_1 to the machine m_0, then, a_2 and a_3 to m_1 and finally, a_4 and a_5 to m_2.The run-time of allocated machines is 250 mins. In a semi-online context, incoming tasks are first gathered and then allocated at the start of the next time period. In the example we consider a time period of 3 seconds. Within this context, the optimal placement shown in 2^{nd} row Figure 4.1 corresponds to 210 mins total run-time. Machine m_0 allocates its first task at 6 seconds while m_1 and m_2 start respectively at 9 and 12 seconds. In the first time period of 3 seconds the system first collects the tasks a_0, a_1, a_2. From 3 to 6 seconds the system solves the placement concerning the tasks received between 0 and 3 seconds. At the same time, it also collects a_3, a_4, a_5 for future placement. At 6 seconds the system implements last processed solution and allocates a_0 and a_2 to m_0 and a_1 to m_1. From 6 to 9 seconds the system decides the placement of a_4, a_5. At 9 seconds it implements the solution by allocating a_4 to m_1 and a_3 and a_5 to m_2. Note that in the semi-online framework the

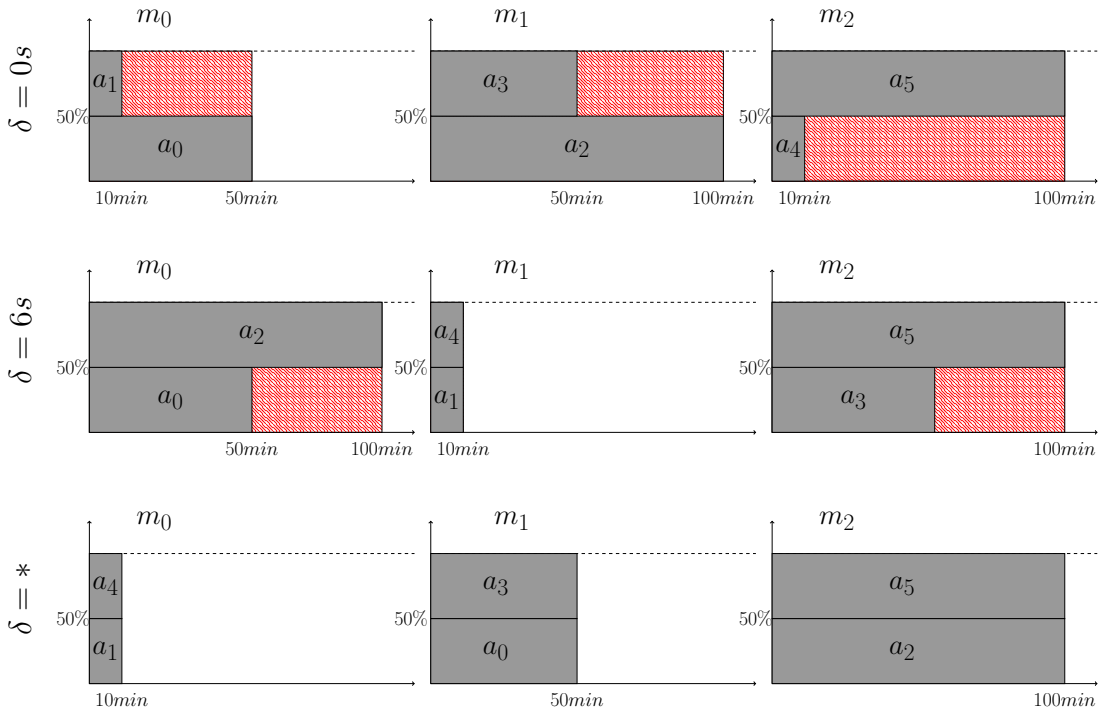


Figure 4.1: Optimal placement considering respectively an on-line, a semi-online (time window of 3s), and a off-line contexts.

waiting time δ between the arrival and the allocation of a task is at most twice greater than the time period of 3 seconds. A small waiting time positively affects on-demand QoS[1].

In the off-line context where all incoming tasks are known before being allocated, the optimal placement shown in 3^{rd} row Figure 4.1 corresponds to 160 minutes of run-time of allocated machines. In this context, the system has decided the placement of the tasks before they arrive, they are allocated as soon as they arrive.

Intuitively, the greater the knowledge of the incoming tasks, the better the consolidation. However, to satisfy on-demand assignment of tasks placement strategies cannot afford to wait too long to have a greater knowledge. By using a semi-online framework we expect to have more information that can lead to better overall assignments while satisfying on-demand QoS.

[1] $*$ = undefined

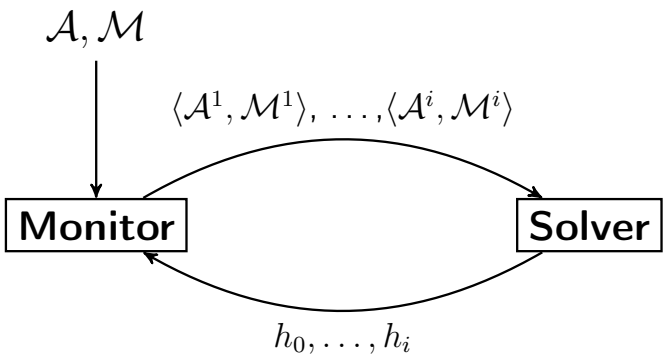


Figure 4.3: The semi-online framework

4.1.1 The Semi-Online Framework

We introduce the semi-online on-demand bin packing problem for which the objective is to minimise the global waste of CPU resources allocated across the pool of machines. The semi-online framework [2] is implemented using two distinct modules as illustrated in Figure 4.3. The first module acts as a monitor and receives the stream of tasks $\mathcal{A}$ to be allocated to the pool of machines $\mathcal{M}$. At each time step i, from the previous placement solutions sent by the solver, the monitor updates and sends back in the set $\mathcal{M}^i$ the information representing the current state the machines. From the stream of incoming tasks $\mathcal{A}$, the monitor also gathers the tasks received during the current time step i into the set $\mathcal{A}^i$ before passing them to the solver.

Finally from the current state of the machines $\mathcal{M}^i$ and newly arrived tasks $\mathcal{A}^i$ the solver builds the corresponding packing problem. The resulting placement solution h_i, mapping each incoming task to a machine, is then sent to the monitor module.

4.1.2 The Monitor Module

The monitor module decomposes time into a sequence of time steps of size tw measured in seconds. Each time step i is mapped to the end time t_i of the corresponding time period. Each task a received during time step i is characterised by an arrival time t_a, an expected duration $\bar{d}_a$ and a required CPU resource q_a. The starting time $\bar{t}_a$ of a corresponds to the end of the next time step after the task has been received, $\bar{t}_a = t_{i+1}$. We implicitly guarantee on-demand placement by requiring the solver to return a consolidation plan within a single time step, which ensures each task will start within $2 \times tw$ seconds of its arrival time.

[2] `https://gitlab.insight-centre.org/mdecauwer/SemiOnlineConsolidation`

Algorithm 3: Monitor

Input: $\mathcal{A}, \mathcal{M}, h_{i-1}$
Output: $\mathcal{A}^i, \mathcal{M}^i$

1 **for** ***each*** $a \in \mathcal{A}, t_a < t_i$ **do**
2 $\quad d_a \leftarrow max(0, \bar{t}_a + \bar{d}_a - t_i)$
3 $\mathcal{A}^i \leftarrow \{(a, d_a, q_a) \mid a \in \mathcal{A}, t_i - tw \leq t_a < t_i\}$
4 **for** ***each*** $m \in \mathcal{M}$ **do**
5 $\quad RunningTasks(m) \leftarrow \{a | \forall j < i, a \in \mathcal{A}^j, h_j(a) = m, d_a > 0\}$
6 $\quad C_m \leftarrow 1 - \sum_{a \in RunningTasks(m)} q_a$
7 $\quad l_m \leftarrow \max(\{d_a \mid a \in RunningTasks(m)\})$
8 $\mathcal{M}^i \leftarrow \{(m, l_m, C_m) \mid m \in \mathcal{M}\}$
9 return $\mathcal{A}^i, \mathcal{M}^i$

Algorithm 3 provides details on how the *monitor* module is implemented. The monitor first updates (resp. initiates) the remaining duration d_a of the tasks already in progress (line 8). Note that the tasks that have completed have a remaining duration of 0. The ones that have not yet started will have a remaining duration initialised after being scheduled. The monitor then gathers the tasks revealed during time step i into the set $\mathcal{A}^i$ (line 3). The tasks that have been placed but that have not yet completed are gathered into the set $RunningTasks(m)$ (line 5). This set builds upon the previous placement solutions h_j, $j < i$. A placement solution h_j, received during time step j, is a mapping from the incoming tasks A^j to the machines in the cluster $\mathcal{M}$.

From the tasks running at step i, the monitor updates the remaining capacity C_m of each machine m (line 6). It also updates the expected remaining run-time l_m of each machine (line 7). l_m represents the maximal expected duration of tasks currently allocated to the machine m, $l_m = 0$ if the machine is currently hosting no tasks. In the end, the monitor sends both the current state of the machines $\mathcal{M}^i$ and the incoming tasks $\mathcal{A}^i$ to the solver (line 9).

4.1.3 The Solver Module

At each time step i the solver is called to solve the On-demand bin packing Problem (ODBP) corresponding to the current state of cluster $\mathcal{M}^i$ and newly arrived tasks $\mathcal{A}^i$. The goal is to return a valid placement h_i of the incoming tasks $\mathcal{A}^i$ on the physical servers $\mathcal{M}$ minimizing the expected run-time of allocated machines. The solver has no further knowledge of subsequent arriving tasks. The tasks for which the placement decision is made during time step i will start running on the assigned machines at the

beginning of the next time step $i+1$. In the following ODBP problem formulation, each $\{0,1\}$ decision variable $x_{a,m}$ denotes the assignment of the task a to the machine $m \in \mathcal{M}$ such that $x_{a,m} = 1$ if task a has been assigned to machine m, $x_{a,m} = 0$ otherwise. The auxiliary integer variable e_m denotes the expected run-time of a machine m. It is entirely determined by the decision variables $x_{a,m}$.

The input data d_a and q_a have the same meaning as before; they represent the expected duration and the CPU requirement of the task a. Similarly, l_m and C_m denote the current maximal expected remaining run-time and the expected remaining capacity of the machine m. We denote by u_m the maximal expected duration of the remaining and the current incoming tasks, $u_m = \max(l_m, \{d_a \mid a \in \mathcal{A}^i\})$. The mathematical model corresponding to the on-demand bin packing problem is as follows:$ODBP(\mathcal{A}^i, \mathcal{M}^i)$:

$$\min \sum_{m \in \mathcal{M}^i} e_m \tag{4.1}$$

$$\text{s.t.} \tag{4.2}$$

$$\sum_{m \in \mathcal{M}} x_{am} = 1 \qquad \forall a \in \mathcal{A}^i \tag{4.3}$$

$$\sum_{a \in \mathcal{A}^i} x_{am} \cdot q_a \leq C_m \qquad \forall m \in \mathcal{M}^i \tag{4.4}$$

$$x_{am} \cdot d_a \leq e_m \qquad \forall m \in \mathcal{M}^i \quad \forall a \in \mathcal{A}^i \tag{4.5}$$

$$x_{am} \in \{0,1\} \qquad \forall m \in \mathcal{M}^i \quad \forall a \in \mathcal{A}^i \tag{4.6}$$

$$e_m \in [l_m ... u_m] \qquad \forall m \in \mathcal{M}^i \tag{4.7}$$

At each time step i the solver aims at minimising the sum of the expected remaining run-times of the allocated machines (4.1). The placement solution returned by the solver enforces the following constraints. Each incoming task is assigned to exactly one machine (4.3). The sum of CPU requirement of incoming tasks assigned to a machine do not exceed the current machine capacity (4.4). The expected remaining run-time of a machine is bound by the largest remaining duration of any incoming task assign to it (4.5) and the maximal expected remaining run-time of the machine constraints (4.7).

A solution of the above mathematical model is a placement reflected in the assignment of the x_{am} boolean variables (4.6). Thus, given a current state of the cluster $\mathcal{M}^i$ and newly arrived tasks $\mathcal{A}^i$, the mapping $h_i : \mathcal{A}^i \mapsto \mathcal{M}$ sent back to the monitor is a function s.t. $h_i(a) = m$ iff $x_{am} = 1$.

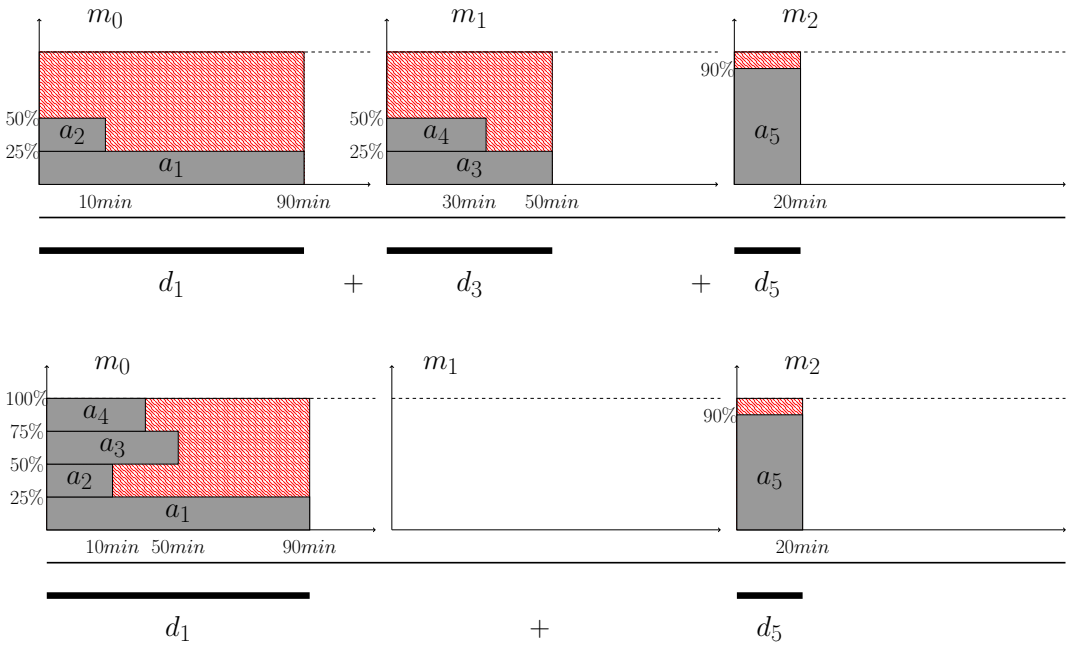


Figure 4.4: Two valid assignments of tasks $\{a_1, \ldots, a_5\}$ to 3 standby machines $\{m_0, \ldots, m_2\}$. All tasks are starting at the current time step t.

4.1.4 Illustrating the Consolidation of Machine Run Times

In Figure 4.4, we consider two placements of five tasks, $\{a_1, \ldots, a_5\}$, with the following duration in minutes (mins): $d_1 = 90$, $d_2 = 10$, $d_3 = 50$, $d_4 = 30$, $d_5 = 20$. The machines, $\{m_0, m_1, m_2\}$, have the same CPU capacity. The tasks have the following CPU requirements (in percentage of the CPU capacity): $q_1 \ldots q_4 = 25\%$ and $q_5 = 90\%$. In the first assignment depicted at the top of the figure, the objective function evaluates the wastage of m_0 to $w(m_0) = 90 - ((25\% \times 10) + (25\% \times 90)) = 65$ minutes of CPU resources. m_1 and m_2 waste, respectively, $w(m_1) = 50 - ((25\% \times 30) + (25\% \times 50)) = 30$ minutes and $w(m_2) = 20 - (90\% \times 20) = 2$ minutes. The total wastage of the first assignment is then $65 + 30 + 90 = 185$ minutes of allocated CPU. Similarly, in the the second assignment of the Figure 4.4, the total wastage is evaluated to $w(m_0) + w(m_1) + w(m_2) = 45 + 0 + 2 = 47$ minutes of allocated CPU. The second assignment wastes fewer CPU resources than the first assignment.

Note that the total running time of active machines in the first assignment is $d_1 + d_3 + d_5 = 160$ minutes. The total running time of active machines in the second assignment is $d_1 + d_5 = 110$ minutes. Implicitly, this example gives us insights into the importance of task duration. It also shows the relationship between the total wastage and the total running time of active machines.

Minimising the global duration of active machines is a difficult task. In [CMO16] where an offline version of ODBP is tackled, the approaches using state-of-the art CPLEX and CP solvers cannot solve the problem optimally for more than a few hundred tasks. In the context of real world on-demand task placement, heuristic approaches are needed to cope with problem sizes reaching thousands of tasks. In the following, we focus our study on heuristic methods minimising the wastage in homogeneous platforms in data centres for minimising the global duration of active machines.

4.2 Packing Heuristics

A range of policies originally developed for the bin packing problem can be adapted to produce solutions to ODBP in a semi-online fashion. The hard constraint on the bin capacities is an aspect usually tackled by these policies such as the family of AnyFit (First, Next, Best, Worst) policies, Sum of Squares and Harmonic heuristics. These heuristics are discussed in [CJCG$^+$13a].

Due to the semi-online nature of the problem at hand, solving policies must only consider information available up to time i when finding an assignment for tasks $\mathcal{A}^i$. An aspect traditionally not handled by bin packing models is the duration for which an item will be consuming resources on its host bin. As shown in Section 4.1, the current expected run-time of machine m is modeled by l_m.

In addition to the policies themselves, the order of tasks to be assigned in any particular time window may significantly impact the solution's quality. The natural order is given by the **LIST** and leaves the tasks $a \in \mathcal{A}^i$ ordered by their arrival time t_a. Alternatively, the order **D** sorts the incoming tasks by decreasing duration $d_a(i)$. Finally, at every time step i, the set of machines is ordered on their l_m values.

First Fit (FF) The FF policy can be applied as-is to the ODBP problem. The tasks in $\mathcal{A}^i$ are in turn assigned to the first machine that has enough remaining capacity. The condition for assigning task a to machine m at time i is thus $q_a \leq C_m$. Assigning a to m updates the remaining capacity on machine m such that $C_m = C_m - q_a$.

Next Fit (NF) The NF policy maintains a pointer to the machine that was last selected to host a task. In turn, each task $a \in \mathcal{A}^i$ will be assigned to the machine currently referenced by the pointer. If there is insufficient remaining space on this

machine (i.e. $q_a > C_m$), NF will move the pointer to the next machine. If no machine in the list of active machines can accommodate the task under consideration (i.e. the pointer reaches the machine it started with), a non-active machine will host the task. The position of the pointer is maintained across the successive optimisation steps.

Best Fit on Requirements (BFR) / Best Fit on Duration (BFD) Since items are characterised by both size and duration, the best fit policy is ambiguous in the context of ODBP. BFR acts similarly to the Best Fit policy for the bin packing problem. For each task $a \in \mathcal{A}^i$, BF selects $m \in \mathcal{M}$ so that the quantity $C_m - q_a$ is minimised. On the other hand, the BFD policy focuses on finding the machine with the closest running time hence minimising the quantity $|l_m - d_a|$. Naturally if no machine in the set of active machines has enough spare capacity to accommodate the task, a new machine is made active and the task assigned to it.

Max Rest - MR The Max Rest policy, also known as Worst Fit acts as the opposite of the BFR policy. Each task $a \in \mathcal{A}^i$ gets assigned to the machine maximising the quantity $C_m - q_a$.

Sum Square - (SS) The Sum-of-Squares algorithm was introduced by János Csirik et al [CJK+99]. Sum-of-Squares uses the notion of the gap of a bin which is its spare capacity. The number of bins with spare capacity g is denoted by $N(g)$. Initially, $\forall g : N(g) = 0$. Then SS assigns an item a of size q_a such that the the quantity $\sum_{1 \leq g \leq B} N(g)^2$ is minimised. Here, B stands for the capacity of the bins. The main intuition behind this algorithm is that is maximises the likelihood of finding a item that *almost* perfectly fits the gap in a bin at any time.

Harmonic - (HA) Lee and Lee [LL85] introduced the Harmonic heuristic for bin packing with the underlying idea being an harmonic partitioning of items and bins on the segment $[0, 1]$ into M families. In our case, the M parameter depends on the number of machines that are hosting tasks at solving time. This partitioning allows us to classify items in an efficient manner. We reuse that idea but instead of partitioning items on their sizes, we partition them on their remaining duration.

4.3 A novel placement policy: First Merged Fit (FMF)

On-line placement policies (cf. Section 4.2) make a clear separation between already allocated machines hosting a set of running tasks and the list of upcoming tasks $\mathcal{A}^i$ that have to be allocated. The policies iteratively allocate tasks to machines, and may miss the opportunity to associate the first two tasks together before assigning them to a machine. This is the main idea behind our algorithm. We propose a flexible approach that does not try to allocate a task to a machine but rather allows each task to be associated to another task or machine.

4.3.1 Illustration

A key concept of the algorithm is to group both allocated machines and arriving tasks under the general notion of *bin*. A bin, thus, represents either a machine currently running, or a set of tasks to be allocated, or both. In Figure 4.5, we illustrate an execution of our approach. The first step is to create a bin for each running machine and each arriving task. Then, the list of bins is sorted according to some criterion. In the example, the order of the machines prioritises the longest remaining running time. At each iteration the algorithm merges the best ranked bin with the next compatible bin in the list.

Two bins are said to be compatible if: (i) at most one of the bins is built from a currently running machine, (ii) if one of the bins is already hosting a task, then merging must be into that bin, without exceeding its capacity; otherwise, merging can be in either bin but must respect that bin's capacity. As an illustration, in Figure 4.5, b_1 is the bin hosting the longest task. Since the second bin is b_2, b_1 and b_2 are compatible, b_2 and b_1 merge into b_1. The merge operation transfers the tasks from b_2 to b_1 and removes b_2 from the queue. No other bins in the queue are compatible with the updated bin b_1. Tasks from the newly created bin b_1 will be allocated to a new machine (m_2). Next up in the list is b_3 which is built from a currently running machine, as is b_4. Consequently, b_3 and b_4 are not candidates for merging. b_3 will be merged with b_5 to form the updated bin $b_3(m_1)$. The algorithm terminates with no new tasks to be allocated.

4.3.2 The First Merged Fit Algorithm

Algorithm 4, *First Merged Fit* (FMF), receives the set of incoming tasks $\mathcal{A}^i$ and the current state of the machine $\mathcal{M}^i$ and returns a valid placement h_i assigning the incom-

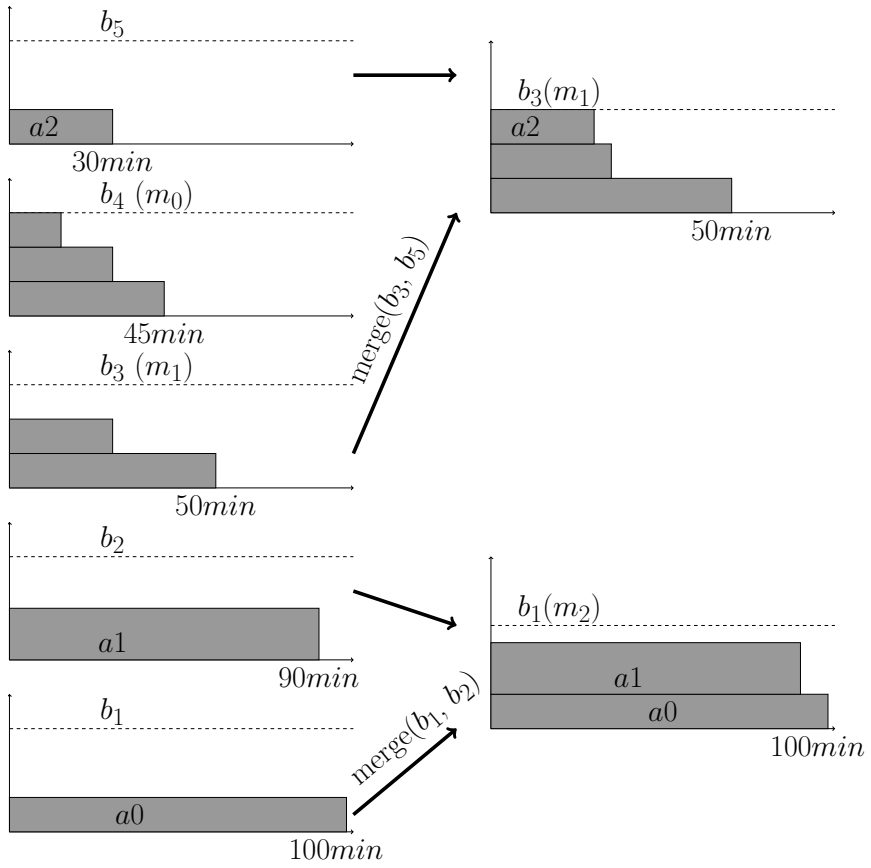


Figure 4.5: A run of First Merged Fit (FMF).

ing tasks to the machines. First, each machine currently running, and each upcoming task is mapped to a bin (line 1). Implicitly, in our approach, a bin is an object that models the expected state of a physical machine that will be hosting the incoming tasks assigned to it. Thus, a bin either corresponds to an already running machine or represents a machine that will start running with its new set of assigned tasks. The function $buildBins$ returns the bins corresponding to each incoming task and each machine currently running ($C_m < 1$), $buildBins(\mathcal{A}^i, \mathcal{M}^i) = \{b \in B \mid (d_b, q_b) = (d_a, q_a), \forall a \in \mathcal{A}^i\} \cup \{b \in B \mid (d_b, q_b) = (d_m, (1 - C_m)), \forall m \in \mathcal{M}^i\}$. For a bin b, d_b represents the expected duration of the task, q_b the sum of CPU requirement. The list of bins B is sorted according to the maximal duration of tasks of each bin (line 2). Each iteration sees the best-ranked bin b_i, line 4, merged with the next compatible bin b_j in the queue (line 7). When the bin is filled, i.e. no further bins can be merged with the current bin, each task allocated to the bin is then mapped to a physical machine for placement. The function $physicalMachine(b)$ returns either the running machine corresponding to the bin or an unassigned machine.

Algorithm 5 gathers the tasks of two bins in one (line 10) and updates the state of the

Algorithm 4: First Merged Fit (FMF)

Input: $\mathcal{A}^i, \mathcal{M}^i$
Output: h_i
1 $B \leftarrow buildBins(\mathcal{A}^i, \mathcal{M}^i)$
2 $SB \leftarrow sortByMaxDuration(B)$
3 **while** SB *is not empty* **do**
4 $\quad b_i \leftarrow pop(SB)$
5 $\quad b_j \leftarrow nextAllocableWith(b_i, SB, \mathcal{M}^i)$
6 $\quad$ **while** $b_j \neq null$ **do**
7 $\quad\quad b_i \leftarrow merge(b_i, b_j, SB, \mathcal{M}^i)$
8 $\quad\quad b_j \leftarrow nextAllocableWith(b_i, SB, \mathcal{M}^i)$
9 $\quad$ **for** ***each*** $a \in b_i$ **do**
10 $\quad\quad h_i(a) \leftarrow physicalMachine(b_i)$
11 return h_i

Algorithm 5: merge

Input: $b_i, b_j, SB, \mathcal{M}^i$
Output: b_r
1 $SB \leftarrow SB \setminus b_j$
2 **if** $b_i \in \mathcal{M}^i$ **then**
3 $\quad b_r \leftarrow b_i$
4 $\quad b_s \leftarrow b_j$
5 **else**
6 $\quad b_r \leftarrow b_j$
7 $\quad b_s \leftarrow b_i$
8 $d_{b_r} \leftarrow max(d_{b_r}, d_{b_s})$
9 $q_{b_r} \leftarrow q_{b_r} + q_{b_s}$
10 $\mathcal{A}_{b_r} \leftarrow \mathcal{A}_{b_r} \cup \mathcal{A}_{b_s}$
11 return b_r

bin accordingly (lines 8-9). In the case of a merge between a bin associated with a running machine and a bin associated with a new machine, the bin receiving the merge is the bin associated with the running machine (lines 3-6).

Algorithm 6 searches the next compatible bin starting from the index of the last visited bin j and iterates over the queue of unvisited bins (line 1). If a compatible bin is found, i.e. the input bin b_i and the visited bin b_j do not both represent running machines (line 2) and the sum of the resources requirement is not excessive, the two bins will be merged (cf. Algorithm 5). If no compatible bin is found, the bin is completed, the sets its allocated tasks will be placed and start running on the corresponding machine at the next time step.

Let $n = |B|$ be the number of bins, i.e. the number of running machine plus the number of upcoming tasks. The complexity of sorting the list of bins is $O(nlog(n))$. For each bin in the list (Algorithm 4 Line 3) we only iterate over the remaining part of the list (Algorithm 6) in descending order. In the worst-case, no compatible bins are found,

Algorithm 6: $nextAllocableWith$

Input: $b_i, SB, \mathcal{M}^i$
Output: b_j
1 **for** ***each*** $b_j \in SB$ *from j* **do**
2 $\quad$ **if** ***not*** $(b_i \in \mathcal{M}^i$ *and* $b_j \in \mathcal{M}^i)$ **then**
3 $\quad\quad$ **if** $q_{b_i} + q_{b_j} \leq 1$ **then**
4 $\quad\quad\quad$ return b_j
5 Return null

so the list of bins does not decrease over the iteration. In this case the complexity is $n \times (n-1)/2$. Overall, the complexity of *FMF* is $O(nlog(n)+n \times (n-1)/2) = O(n^2)$.

4.4 Local Search

The previously described policies can be used to produce valid solutions heuristically in a rather short amount of computational time. In a real operational setting one could use the time left in the window to try to converge to better solutions using techniques such as neighborhood search. The underlying idea is to build a MIP model capturing the decision problem local to a time window. This local problem is composed of the state of the system and the list of incoming tasks for which an assignment is expected. The various policies are used to produce a feasible solution (incumbent) in turn provided as a first assignment to the complete solver (CPLEX). The complete solver is then allowed the time left to explore further solutions.

CPLEX was tuned to use a neighborhood search technique (RINS) method described in [DRP05] as a black box. RINS exploits information contained in the linear relaxation of the MIP model. RINS can be thought as an anytime approach to neighborhood search in the sense that it always yields the best feasible solution found so far.

4.5 Empirical Analysis

In these experiments, we first compare the performance of the workload consolidation of the incoming tasks as reported in the trace (i.e. we assume we know the duration of each task when it arrives). Section 4.5.1 compares the total allocated resources of the different approaches for the period one week of incoming tasks. Section 4.5.2 analyses the resource consumption of the policies during high demand of tasks placement. Section 4.5.3 measures the resource usage when varying the time step duration. Section 4.5.4 relaxes the hypothesis on full knowledge of task duration and measures the resource usage when varying the uncertainty of task duration. Section 4.5.5 compares the time performance. Last, Section 4.5.6 summarises the resource usage and the time performances of the approaches.

4.5.1 Overall Allocated Resources

Figure 4.6 shows the solution quality of the different approaches by comparing the total allocated resources over time. Using the simulation framework described in Section 4.1.1, we measured the performance of heuristics in terms of cumulative allocated resources (y-axis) as the simulated time progresses (x-axis).

The allocated resource is measured in terms of run-time of allocated machines. The time step duration is two seconds. Figure 4.6 a) shows the evolution of the total resource usage of each policy. Figure 4.6 b) shows the evolution of the total resource usage when the remaining time left is used to improve the solution returned by each policy using a neighborhood search heuristic [DRP05] with the CPLEX solver.

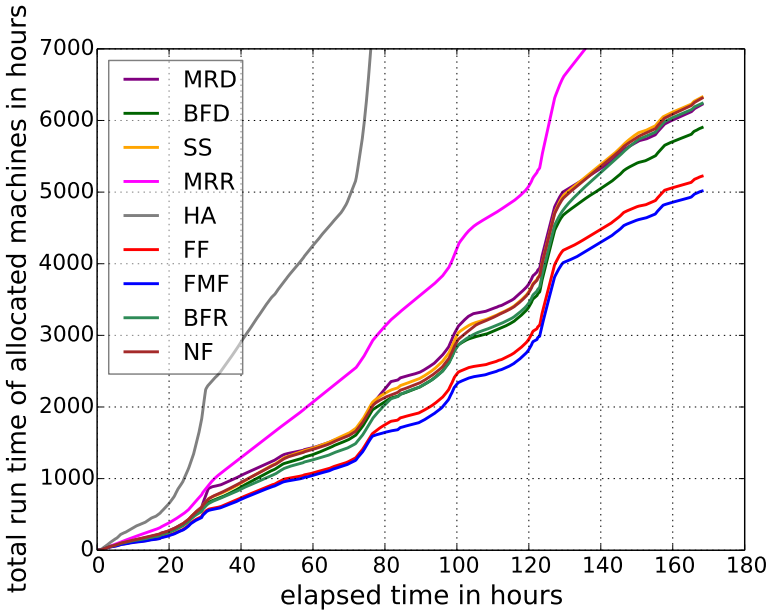
The total resource allocated over time interleaves steady growths and levels. Steady growth corresponds to time periods when few tasks arrived per time-window. Levels correspond to arrivals of peaks of incoming tasks. In periods of steady growth all the approaches slightly increase the total allocated resources. The gap in resource utilisation of the approaches slightly increases after each change of level. At these times, the placement of tasks becomes more challenging and the difference between the solution quality of the policies increase accordingly. We analyse in details the arrival of peaks of tasks in Section 4.5.2.

The best placements policies are FMF followed by FF then BFD. The placement policies HA and MRR waste significantly more resources and exceed the resource limit shown in the Figure 4.6a. When the solution of each policy is enhanced by the neighborhood search, Figure 4.6b, less efficient policies significantly improve their resource usage and narrow the gap between the best policies. Note that the use of neighborhood search marginally improves the resource usage of the best policies.

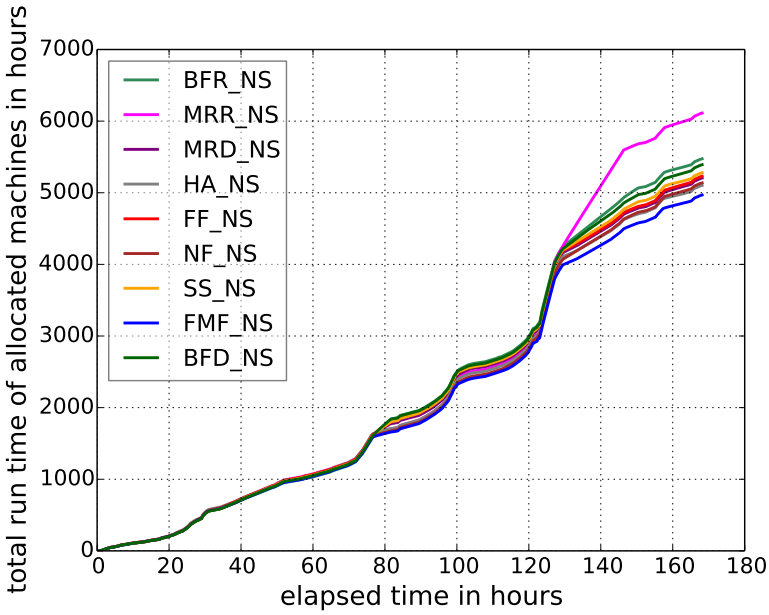
In Table 4.1 we present the total resources allocated by the different approaches after seven days, i.e., 168 hours of workload consolidation. Compared to the leading standard bin-packing heuristics, FMF consumes 4% less resource than FF and 15% less resource than BFD. Compared with the most inefficient policies, FMF saves 1.75 and 3.6 times more resource than MRR and HA.

Table 4.1: Total run-time of allocated machines (in hours) of the placement policies

Approaches:	FMF	BFD	BFR	FF	HA	MRD	MRR	NF	SS
Policy:	**5016**	5902	6239	5223	18232	6225	8786	6313	6327
Policy+NS:	**4969**	5392	5474	5231	5102	5205	6113	5138	5279



(a) Policies

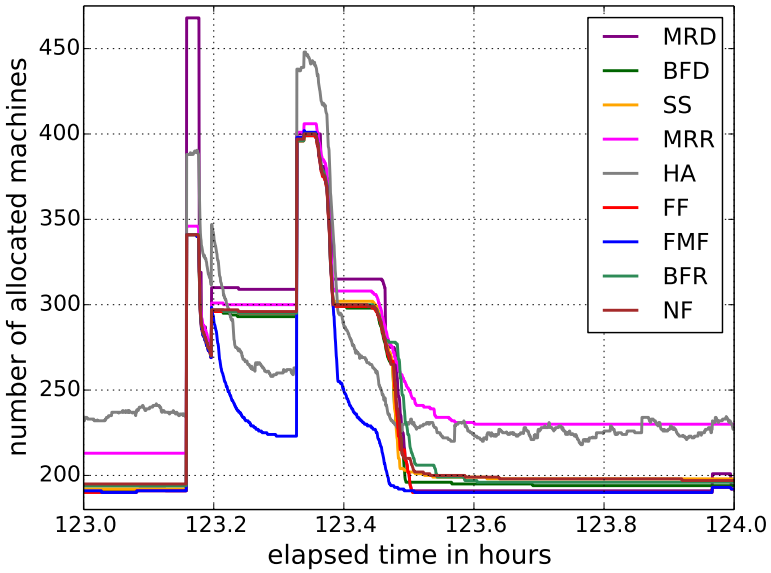


(b) Policies + neighborhood search

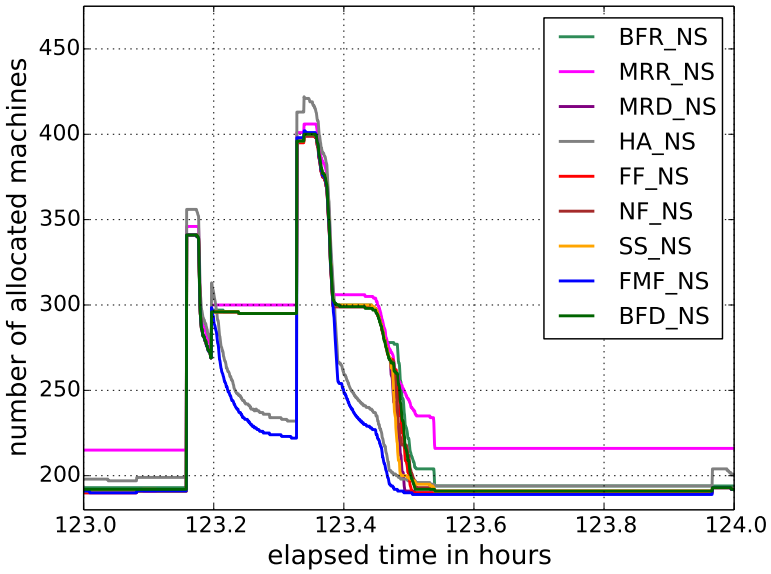
Figure 4.6: Total allocated resources over the time per policies against elapsed time.

4.5.2 Resource Allocation During Peak Activity Periods

We analyse the resources allocated and released after incoming peaks of tasks. Figures 4.7 a) and 4.7 b) compare the behaviour of the approaches at the arrival of three peaks of 3293, 959 and 3899 tasks received at different time steps. During the 123^{rd} hour 10246 tasks have to be placed, the three noticeable peaks represent 80% of the



(a) Policies



(b) Policies + neighborhood search

Figure 4.7: Number of allocated machines over the peaks of the 123^{rd} hour.

incoming tasks. We thus focus on that specific hour of the simulation for the sake of illustrating differences in allocation policies under stress load. In both figures, the x axis corresponds to a specific time span of one hour, i.e. 1800 time steps of 2 seconds of the 123^{rd} hour of the simulation of task arrivals. The y axis describes the number of allocated machines.

In Figure 4.7a, at the arrival of the first noticeable peak of tasks, all the approaches perform similarly. They allocate new machines to cover the peak requirement. The

Table 4.2: Run-time of allocated machines above 180 machines during the 123^{rd} hour.

	Policy:	FMF	BFD	BFR	FF	HA	MRD	MRR	NF	SS
$tw = 2$	Alone	**35.8**	51.8	54.9	49.7	74.7	57.2	76.8	55	54.1
	+NS	**35.3**	51	53.5	50	43.2	49.5	69.9	50.6	50.3
$tw = 30$	Alone	**35.9**	52.4	52.2	48.6	71	53.5	75.5	53.2	53.4
	+NS	**36.4**	48.4	50.2	48.1	42.1	48.5	67	48.6	48.8

same behaviour can be observed at the arrival of the second and the third noticeable peaks. The quality of the tasks consolidation methods can in fact be observed after the peaks when the CPU resource is gradually released by the finishing tasks. Placement policies from the related work waste significantly more resource than FMF. At the end of the first and the third peaks, HA allocates up to 40 more machines than FMF while the others placement policies allocate up to 80 more machines than FMF.

Figure 4.7b shows the resource allocated by the policies enhanced by neighborhood search heuristics. Only the less efficient placement policies HA, MRD and NF show a clear benefit of using the neighborhood search heuristic to improve their solutions. At the arrival of the first peak of incoming tasks the number of allocated machines used by MRD is significantly reduced when using neighborhood search and passes from more than 450 machines to less than 350 machines. The resource consumption of HA enhanced by neighborhood search is also significantly reduced. It remains constantly close to FMF along the hour. For the other methods, the neighborhood search approach does not improve the solution returned by the policies. In these cases, the solutions returned by the policies are already good. The neighborhood search does not have the time to improve the placement within the time step of two seconds. In Table 4.2, 4^{th} line, we also check the performance of the enhanced policies when the duration of each time step has been increased to 30 seconds. The results show that with a longer time step duration, the gain in resources utilisation is not significant for the policies already proposing an efficient consolidation.

Note that at the beginning and at the end of the 123^{rd} hour more than 180 machines remain allocated. Table 4.2 shows the extra resource above 180 machines allocated during this hour. In the first part of Table 4.2, we show the extra resources allocated when the time step is 2 seconds. To cover the peaks of incoming tasks of the 123^{rd} hour, FF allocates 40% more extra resources than FMF. The other policies, such as BFD and HA, allocate between 40% and 200% more resources than FMF. Table 4.2, 2^{nd} line, shows that HA enhanced by neighborhood search approach drastically reduces the resource consumption by 42%. MRD and NF are improved by approximately 8%. FMF, BFD, BFR and FF show similar resources consumption enhanced or not by

neighborhood search. In these cases, the policies solutions are already efficient, the remaining time dedicated to the neighborhood search heuristic to improve the polices' solutions show no benefit. Note that HA enhanced by a neighborhood search heuristic returned better results than BFD, BFR, FF. Implicitly it seems that the local minimum found by the neighborhood search starting from HA solutions is better than the local minimum found when starting from BFD, BFR and FF solutions.

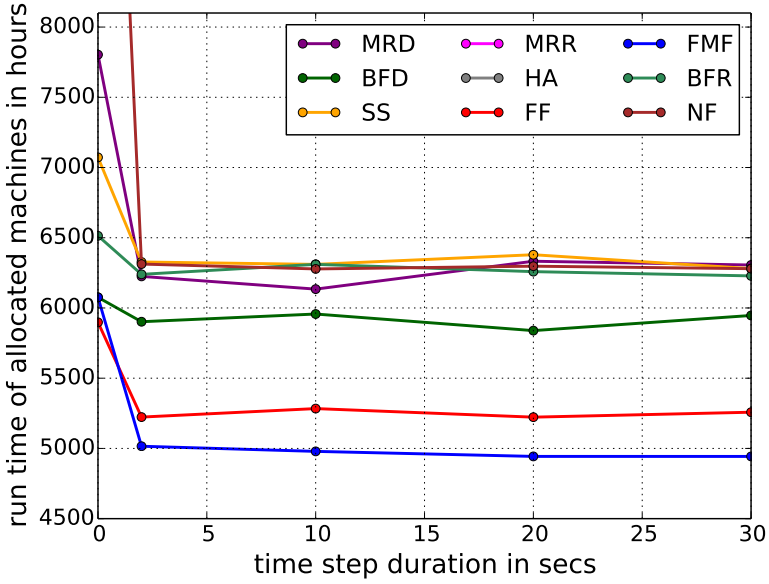
In the second part of Table 4.2, we show the extra resources allocated when the time step is 30 seconds. In this case all the policies slightly improves their resource consumption except FMF, BFD, FF, showing similar results. Using a neighborhood search heuristic drastically improves the allocated resources of HA and more reasonably the resources consumption of MRR, NF, SS, BFD, and BFR. FMF and FF return similar resource consumption than before.

In summary, during a period high resource demand, the placement policies may pay the price of an inefficient placement after the peaks. As a result when the short duration tasks end an unnecessary amount of machines remain active to execute longer time duration tasks. These tasks should have been placed in different machine earlier. FMF is showing a more efficient usage of the allocated resource. This behaviour can be explained by the fact that FMF takes full advantage of the tasks ordering based on duration compared to some other policies such as BFD or MRD. In addition, FMF exploits the opportunity to merge incoming tasks before allocating them to a machine.

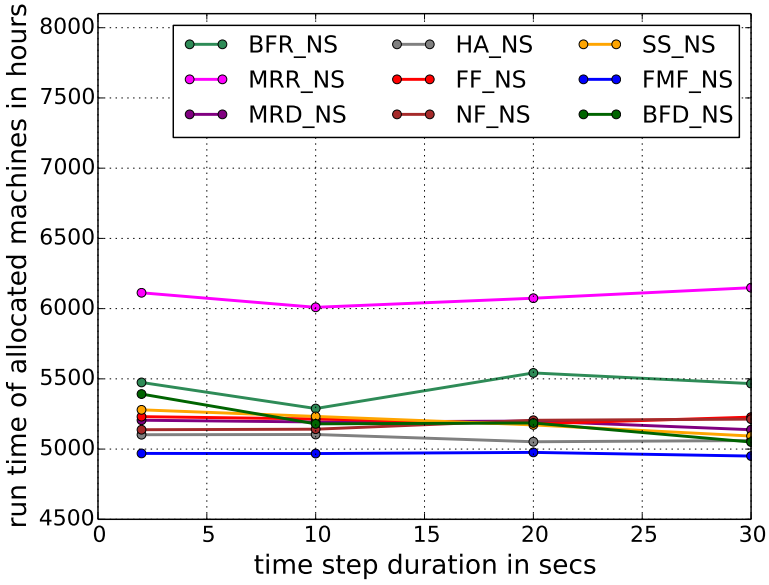
4.5.3 Resource Allocation under Varying Time Step Duration

Figure 4.8 shows the evolution of the resource usage of the different approaches when increasing the time step duration. The resource usage (y-axis) is expressed in run-time hours of allocated machines. The time step duration (x-axis) is expressed in seconds. Note that increasing the time step duration implicitly decreases on-demand QoS ensuring real time placement of tasks. In an on-line context each task has to be placed as soon as it arrives in the system. In the experiments we simulate the on-line context by considering an ordering heuristic based on the arrival time of the tasks. The result of the on-line simulation is shown by the time step duration 0s. The benefit of the semi on-line context described in this study is shown from the time step duration 2s to 30s. Above 30 seconds we consider the On-demand QoS not fulfilled.

In Figure 4.8 a) each policy shows a noticeable improvement of the resource usage from the on-line context, i.e. a time step duration of 0s, to the semi-online context, i.e. a time step duration of 2s. Within the semi on-line context, i.e. from a time step of 2s



(a) Policies



(b) Policies + neighborhood search

Figure 4.8: Allocated resources per policies when increasing time step duration

to a time step of 30s, the resource usage slightly decreases, or remains stable, as the time step duration increases. FMF always shows the best resource utilisation when increasing the time step duration, and is closely followed by FF. Figure 4.8 b) shows the evolution of the resource usage of the placement policies enhanced by the neighborhood search. Compared with the solution returned by policies, the policies enhanced by neighborhood search show a noticeable improvement in the resource usage. FMF and FF are the exceptions, the resource usage remain stable.

Table 4.3: Resource utilisation (in hours) for time step duration 0, 2 and 30 seconds

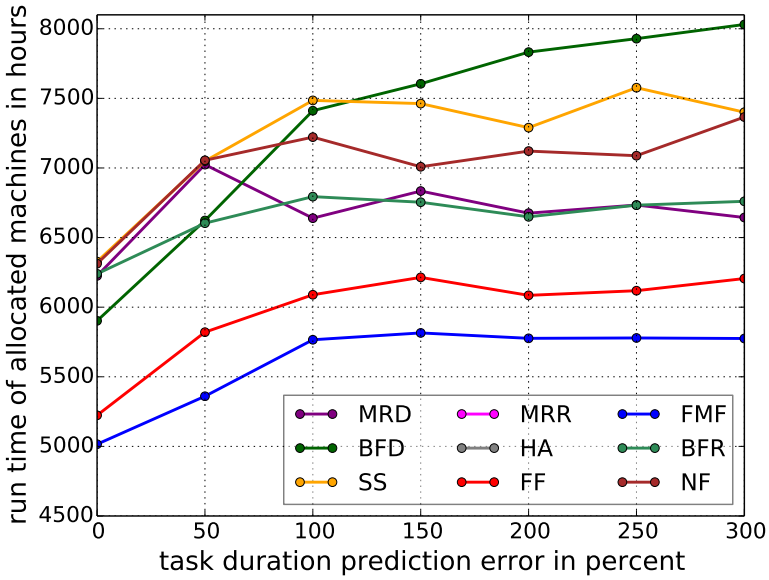
Policy	tw	FMF	BFD	BFR	FF	HA	MRD	MRR	NF	SS
Alone	0*(s)	6076	6076	6514	**5898**	18493	7803	9643	13032	7072
	2(s)	**5016**	5902	6239	5223	18232	6225	8786	6313	6327
	30(s)	**4943**	5947	6228	5258	14255	6306	8648	6280	6280
+NS	2(s)	**4969**	5392	5474	5231	5102	5205	6113	5138	5279
	30(s)	**4950**	5050	5466	5228	5061	5138	6149	5213	5093

Table 4.3 sums up the total resource usage at the specific time duration of 0s, 2s and 30s. In the online context (time step duration = 0), FMF is similar to BFD. From time step duration = 0s to 2s, NF, MRD and FMF respectively reduce the allocated resources by 50%, 20% and 17%. These improvements represent the best gains among the policies for this variation of time step duration. Then, only HA shows a significant decrease in the resource consumption and allocates 22% less resources from a time step duration of 2 to 30 seconds. FMF only reduces its allocated resources by 1.5%. When it comes to the policies enhanced by neighborhood search, only BFD shows a modest decrease in the resource utilisation of 6%. The other enhanced policies keep similar number of allocated resources. Enhanced policies such as MRR+NS, NF+NS slightly increase the resource consumption when passing to a time step duration of 30s. These cases are counter-intuitive since the same methods give better results in a less informed context. In the general case, solving a succession of locally optimized problem leads closer to the global optimal. However it is not always the case. In our experiments the cases of MRR+NS and NF+NS remain marginal.

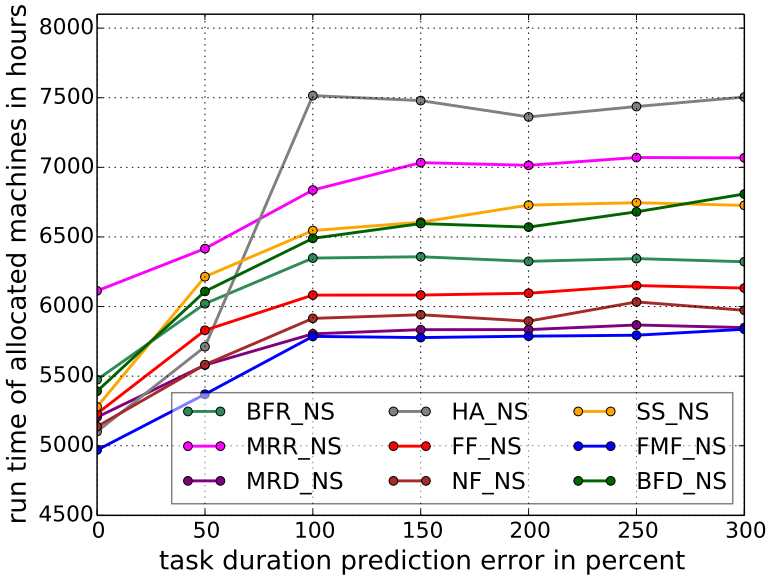
4.5.4 Resource Allocation under Uncertain Task Duration

In this section we relax the assumption on full knowledge of duration and measure resource usage when varying the uncertainty of task duration. In our experiments, given a task duration prediction error of $x\%$ and a task duration d recorded from the data centre traces, the policy is sent the expected duration in $[d \times (1 - x/100), d \times (1 + x/100)]$ for the placement. The simulator updates the expected duration as the time passes, but effectively deallocates a task only when its real duration ends.

Figure 4.9 show for each policy the evolution of the allocated resource when increasing the task duration prediction error. Both type of approaches, i.e., policies (Figure 4.9a) and policies enhanced by neighborhood search (Figure 4.9b), follow the same pattern. First, the resources allocated by the approaches increase linearly as the error prediction increase from 0% to 100%. Then, from 100% the resources allocated remain constant or slightly increase. Within the approaches based on policies, FMF is noticeably better



(a) Policies



(b) Policies + neighborhood search

Figure 4.9: Allocated resources while increasing tasks duration uncertainty

and shows better resource utilisation. The second best is FF while the other policies consume significantly more resources. The policies HA and MRR are outside the scope of the figure, their performance are shown in Table 4.4. Within the approaches using neighborhood search, FMF+NS remains better for prediction error between 0% and 100%. Then, it is closely followed by MRD+NS. Here MRD takes a clear advantage of the neighborhood search heuristic to improve its resource consumption. It is also the case for HA+NS and MRR+NS that now appear in the scope of the figure.

Table 4.4: Total run-time of allocated machines (in hours) of the placement policies tw=2

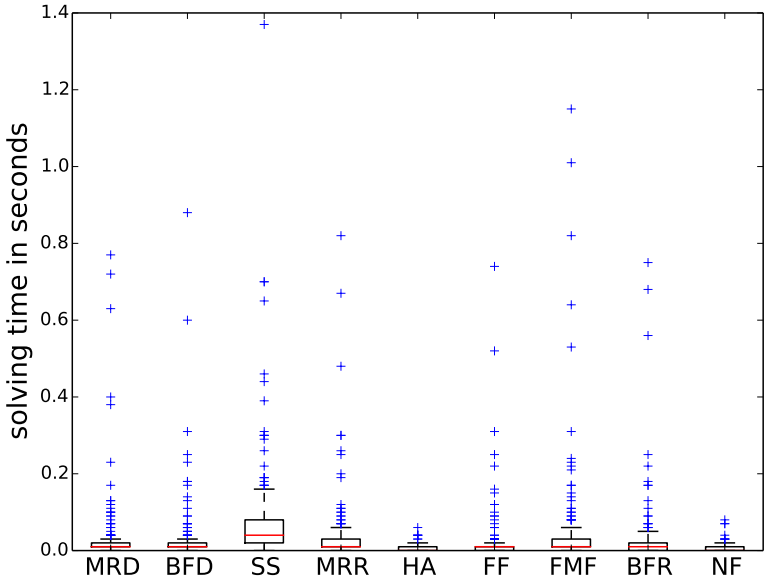
Policy	err	FMF	BFD	BFR	FF	HA	MRD	MRR	NF	SS
Alone	0%	5016	5902	6239	5223	18232	6225	8786	6313	6327
	100%	5766	7411	6794	6089	21181	6639	10590	7222	7485
	300%	5775	8031	6760	6206	21271	6644	10420	7366	7401
+NS	0%	4969	5392	5474	5231	5102	5205	6113	5138	5279
	100%	5786	6490	6348	6082	7515	5804	6837	5915	6546
	300%	5837	6809	6322	6133	7505	5849	7068	5973	6727

Table 4.4 sums up the two patterns followed by the approaches when increasing the task duration prediction error. For a prediction error of 100%, the policy FMF wastes 5% less resources than FF and between 13% and 23% less than the other policies. MRR and HA for that consumes 1.8 and 3.7 times more resources than FMF. For a prediction error of 300%, the policy FMF wastes 7% less resources than FF and between 13% and 28% less than the other policies. Enhanced policies using neighborhood search drastically reduce the resource usage of all policies except FMF+NS and FF+NS.

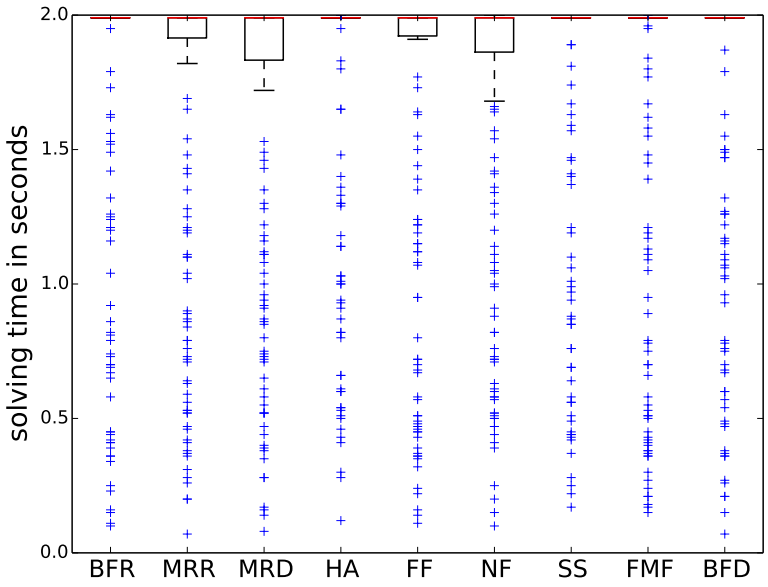
In summary we have analysed the behaviour of the different approaches when increasing the uncertainty of the task duration prediction error. Here again enhancing FMF or FF by a neighborhood search does not improve the resource consumption, and confirms the quality of the the solution returned by these policies. The other approaches clearly benefit from a neighborhood search and reduce the gap with FMF.

4.5.5 Real-time Placement of Incoming Tasks

Figure 4.10 compares the solving time quartiles of the different policies (Figure 4.10a) and the policies enhanced by neighborhood search (Figure 4.10b). Here we consider only the solving time for placement problems having more than 100 incoming task during a time-step during of 2 seconds. For each approach is shown the solving time quartiles using box plots. In this plot, the boxes contain half the solving times. The median solving is shown with the red segment and the blue dots are outliers. In Figure 4.10 a) the approaches based on policies only show an almost instantaneous median solving time that does not exceed 0.1 second. The highest point at the top of each policy denotes the time for solving the largest peak of thousands tasks. The largest peak is solved in 1.2s and 1.4s for FMF and SS, and less than 1 seconds for other methods. NF always answers instantaneously. The NF policy only pays the cost of sorting the task by duration and then assign them as they come to the next available machine.



(a) Policies



(b) Policies + neighborhood search (NS)

Figure 4.10: Solving time (in seconds) of the placement policies when number of incoming tasks > 100.

Even if FMF is able to allocate new incoming VMs into a new machines without the need to iterate over all the allocated machines, here FMF pays the price for rearranging in one list both the incoming tasks and the currently allocated machines. Nevertheless, since the worst solving time remains below the time step duration FMF is able to satisfy the on-demand QoS enforce by the semi-online framework. In contrast, in Figure 4.10 a), the approaches based on policies plus neighborhood search hit the

time limit of 2 seconds corresponding to the time step duration. Implicitly, the solver driven by a neighborhood search tries to improve the given policy solution till the end without proving the optimality. As noticed before, increasing the time step duration from 2 secs to 30 secs drastically improves the solution found by less efficient policies but does not improve the resource usage of FMF. Consequently FMF+NS pays an unnecessary penalty in terms of computational time.

In summary, with a time step period of 2s, all the heuristics are able to place the incoming tasks instantaneously and within the time limit. Only NF Heuristic shows instantaneous placement time in period of peaks of incoming tasks. In this case FMF shows the 2nd worst placement time but remains below the time limit. This modest time performance is compensated by a better resource usage.

4.5.6 Comparing Policies

In this section we summarise the performance of the approaches FMF, FF, BFD, MRD and NF through the radar plot shown in Figure 4.11. The performances of the policies are compared in term of resource utilisation (axes A, B, C, D, E) and time performance (axes F, G). For a policy, the closer to the border, the better the policy is performing. At a glance, we can see that FMF is the best or equal best policy on all axes except one (and that axis, showing solving time within the time window, does not affect the quality of the result).

Axis - A. Resource utilisation in the online context.

Axis A represents the total resource utilisation of the the policies in the on-line context (cf. Section 4.5.3). In this context FF shows the best resource usage. It is followed by FMF and BFD.

Axis - B. Resource utilisation in a semi-online context.

Axis B shows the benefit of the semi-online context. Compared with the on-line context it guarantees a placement within a 2 second time-step. In this case, all policies noticeably improve their resource usage compared with the online context. The FMF becomes the best policy and shows an improvement of 17% of its resource utilisation. Even if NF shows the less efficient consolidation it is this policy that benefits most from the semi-online context and wastes 50% less resources. From a time step of 2 second to a time step of 30 seconds the policies do not improve significantly their solution quality (cf. Section 4.5.3).

Axis - C. The impact of neighborhood search.

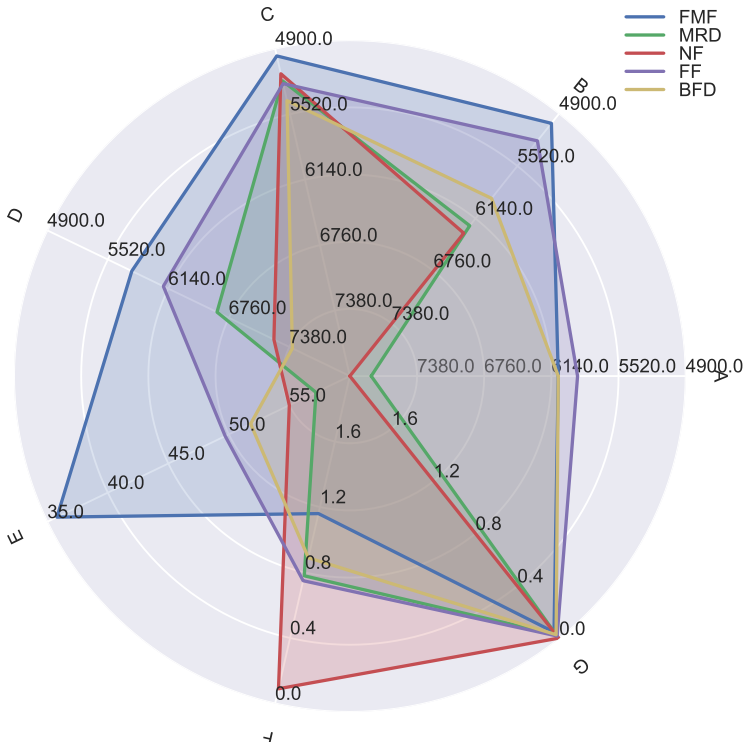


Figure 4.11: Performance of the policies while varying different parameters

Axis C shows the benefit of enhancing the policies with a neighborhood search heuristic in the semi-online context of 2 seconds time-step. Compares with single policies, the benefit of neighborhood search is more noticeable for MRD and NF the less efficient policies that strengthen the gap between FMF. FMF shows the best resources utilisation but it only improves its resource utilisation by 1.5%. This improvement does not change when more time is dedicated to the neighborhood search (cf. Section 4.5.3).This confirms the quality of the solutions returned by FMF.

Axis - D. Resiliency against task duration uncertainties.

Axis D represents the total resource utilisation in a semi-online context with 2 second time-step and an uncertainty of task duration of 100%. Due to the uncertainty of task duration all policies become less efficient and waste more resource. However, FMF shows the best resource consolidation even if it degrades its resource utilisation by 15%. More importantly the wastage is limited and remains stable as the uncertainty in the task duration goes from 100% to 300% (cf. Section 4.5.4).

Axis - E. Activity Peak Behaviour

Axis E represents the extra resource consumes during the 123^{rd} hour at peaks of incoming tasks. This time period represents a more challenging placement

problem since the policies have to deal with a large number of tasks in short delay. Here, FMF clearly outperforms the other policies. The second best policy allocates up to 40% more resource than FMF as seen in Section 4.5.2.

Axis - F and G. Algorithm solving time performance.

The last axes show respectively the maximal solving time (F) and the median solving time (G) for time step receiving more than 100 incoming tasks. The maximal solving time corresponds to the placement of a peak of thousands tasks. Here NF answers almost instantaneously. FMF show the slower solving time even if it remains below 2 seconds (Axis F). In the average case all policies are qualified as real-time approaches (Axis G and Section 4.5.5).

4.6 Conclusion and Limitations

Workload consolidation is a way to reduce the wastage of resources by clustering tasks together on a subset of physical machines. In the literature many successful approaches have studied the problem of workload consolidation from different perspectives. In this study we tackle the challenge of workload consolidation in the context of on-demand resource allocation where data centres want to guarantee real-time allocations of users' tasks. While most of the approaches have envisaged on-line consolidation policies, placing one task at a time, or batch consolidation optimisation, we consider the workload consolidation in the context of semi on-line optimisation. In this new context, we introduce a novel approach that benefits from the short period time windows to take more informed decisions while satisfying real-time requirement of on-demand placement QoS.

We have introduced a model allowing us to reason about the dynamics of the problem. We presented bin packing-inspired heuristics (FF, NF and BF) along with our ad-hoc algorithm (FMF) that implement semi-online workload consolidation by locally (in time) minimising resource wastage.

We have seen that our algorithm, FMF, outperforms bin packing-inspired heuristics on the problem as we have formulated it. After one week of workload consolidation, FMF saves up to 40% more resources during periods of high resources demand than the best adapted heuristics enhanced with neighborhood search. The gap between FMF and the other approaches is more visible in period of peaks of incoming tasks. Moreover, FMF also shows the best resource utilisation performance when increasing the uncertainty of task duration or varying the time period of time step. Even if a better resource

usage comes at the cost of a more time consuming approach, FMF is able to guarantee on-demand QoS.

The operational setting explored in this chapter abstracts some challenges quite often faced by cloud providers. It has been reported in the literature that, at decision time, the incoming workload is not well characterised. This chapter captures the challenge of missing information regarding task duration. In the next chapter we tackle the lack of information regarding tasks in terms of CPU and RAM requirements.

Chapter 5

Online Consolidation with Uncertain Task Sizes

Summary. *In this chapter we are interested in dealing with uncertainties linked to tasks resource requirements. Accounting for a more complex consolidation environment, we develop in this chapter a methodology consisting of three main modules: i) a prediction module that forecasts the maximum resource requirement of a task. ii) a scheduling module that efficiently allocates tasks to machines; and iii) a monitoring module that tracks the levels of utilisation of the machines and tasks, and can evict one or more tasks from the machines for rescheduling if required.*

This chapter finds its motivation in cloud operational settings in which the resource requirements of tasks are estimated rather than fully known at decision time. Facing incomplete information, a rather large discrepancy can be seen between maximum resources expected to be consumed by tasks as specified by users and those that are actually utilised during their lifetime. This is shown to be the case for the data used in this dissertation. Please refer to Section 2.3.5 for a detailed analysis. We focus our optimisation models on the description of the real-world cloud system provided by Google in 2011 [RWH11, RWH12, ARA14].

As a first contribution, we are interested in estimating the peak resource requirements for tasks incoming in the cloud system. Our claim is that it is possible to leverage standard machine learning techniques to produce an estimation of the peak resource consumption of tasks improving on the estimation provided by the users. In turn these forecasts can be used by an online scheduling policy in order to maximise the overall

utilisation of the cluster while minimising the mean waiting time of tasks. The underlying idea is to use more accurate information at scheduling time to reduce as much as possible resource over-provisioning at a level of a machine without sacrificing Quality of Service.

Unlike other traditional scheduling problems [LYQ06], the actual utilisation of the machines cannot be known at decision time. The actual consumption of resources are varying as the tasks are processed. Therefore, the actual utilisation of the machines can only be computed for the time that has already passed, that is, after making the scheduling decisions. A significant body of literature in data centre scheduling is concerned with tasks that can be preempted or paused and migrated on a different machine [VGC+13, VdBVB11]. The problem tackled here does not admit such properties. In the problem at hand, an external mechanism is responsible for *evicting* one or more tasks from an overloaded machine in order to guarantee quality of service for the remaining tasks on the same machine. Those evicted tasks are to be resubmitted for scheduling and processed on a machine from the beginning. In addition, the resource requirements and the duration of tasks can vary significantly, and the number of tasks arriving at any time-point could be tens of thousands. The challenge is to efficiently solve this highly dynamic multi-dimensional online resource constrained scheduling problem consisting of heterogeneous machines and tasks. For this purpose, it is necessary to mitigate the effects of having uncertainties.

We study the behaviour of classical scheduling policies in this highly dynamic and uncertain environment. Our focus is laid on the two aforementioned criteria i.e., maximising the overall system's utilisation and minimising a metric reflecting the average waiting time of tasks in the system. We then design two algorithmic enhancements over simpler policies and show how one can benefit from them. Naturally, since the actual requirements of tasks are uncertain at decision time, we study the impact of uncertain information used by the simple policies in contrast to more advanced scheduling policies. A last aspect discussed in this chapter is related to the eviction mechanism. Beyond the tasks' scheduling policies, the implementation of the eviction mechanism proved to notably influence the performance of the scheduler. The eviction policy implemented in Google's scheduler selects tasks to evict from an overloaded machine based on tasks priorities [RWH11]. We claim that this mechanism can drastically impact the quality of the schedule and explore alternative *eviction policies*.

We provide a method for efficiently scheduling tasks to machines without precise knowledge of task resource consumption and arrival time, given the user expectations on resources requirements of their tasks. The method includes three modules: **predic-**

tion, **monitoring** and **scheduling** allowing us to improve the performance of a large-scale cloud computing system. We argue that these modules are necessary in order to efficiently assess how well-utilized a cluster will be, based on historical data and characteristics of the cluster. We present the results from three different predictors in order to understand the benefits of more accurate resource predictions on the scheduling process. In particular, we apply two machine learning techniques, namely multiple linear regression and random forest [Bre01] to learn a statistical model predicting resource requirements of tasks more accurately than the users would. In addition, we analyze an ad-hoc predictor that assigns to every task a fixed percentage of the user-defined limit. Moreover, the scheduling process requires that we constantly monitor the state of the machines in the cluster as well as the current utilisation levels for each task. This information is valuable in order to periodically update the prediction models. Furthermore, the scheduler also needs to know the status of the machines before deciding where to schedule each task. Finally, we provide a mathematical formulation of the scheduling problem and propose an adaptive and scalable greedy method for allocating tasks to machines.

The remainder of the chapter is organised as follows. Section 5.1 introduces our methodology, including a detailed description of all of its modules in sections 5.2, 5.3 and 5.4. Section 5.5 presents the evaluation and results of the proposed approach.

5.1 Methodology

The main objective of our proposed method is to build a framework in which one can design and evaluate policies for the online scheduling problem at hand. To evaluate our approach we simulate the dynamics of an actual cluster of machines. The goal is to perform efficient online scheduling of tasks on machines subject to operational constraints, without prior knowledge of each task resource consumption and arrival times. As input, we are given, a set of jobs composed of tasks, with their attributes, and the user's expectations on resource requirements along with custom predictions.

The performance of the scheduling policies will be studied both with an owner-centric objective function (i.e. maximizing utilisation of the cluster) and in a user-centric fashion (i.e. minimising waiting time). Both objectives will be defined more formally in Section 5.3. The suggested approach is build around 3 modules briefly introduced as follows:

Predict

This module is responsible for producing estimates on the maximum resource consumption for each task. For this purpose, we use two machine learning algorithms (e.g. multiple linear regression and random forest) and an ad-hoc technique. Section 5.2 discusses these technique in more details.

Schedule

This module aims to schedule tasks on machines, while trying to optimize both machines' utilisation and tasks' waiting time. A constraint optimization problem is formulated and a greedy method is used to decide how the tasks should be scheduled. Section 5.3 formally defines the problem, the objective functions and introduces algorithmic components derived to solve the problem.

Monitor

This module constantly monitors the resource consumption of the tasks as well as the machines, in order to rapidly detect over utilisation of any resource in any machine and evict tasks if required. See section 5.4 for further details.

Figure 5.1 presents an overview of the proposed methodology. The methodology starts with a *queue of tasks* waiting to be scheduled. The tasks are added to the queue as soon as they arrive and the goal is to empty the queue as soon as possible in order to minimise task waiting time. Furthermore, given the characteristics of the tasks, the *predictor* returns the predicted maximum resource consumption in terms of CPU and RAM. This information is then passed on to the *scheduler* which will, in turn, select a machine and a starting time for the task to be processed. Finally, we implemented a *monitoring* module that throughout different events, updates the levels of utilisation of the machines and tasks, completion time of tasks, resource violation from the tasks or over utilisation of the machines. When an event occurs, this information is logged for statistical purposes, and is read by the scheduling module. Moreover, information is also sent periodically to the predictor model in order to improve its accuracy. This module is also in charge of triggering task evictions. Further details on these modules are presented in the next sections.

5.2 Prediction Module

The prediction module aims to estimate the maximum resource consumption of the tasks in terms of CPU and RAM. We denote the predicted maximum CPU and RAM utilisation of a task by $predCPU$ and $predRAM$ respectively. Before discussing the predictor techniques used in this work, it is important to first describe the input attributes,

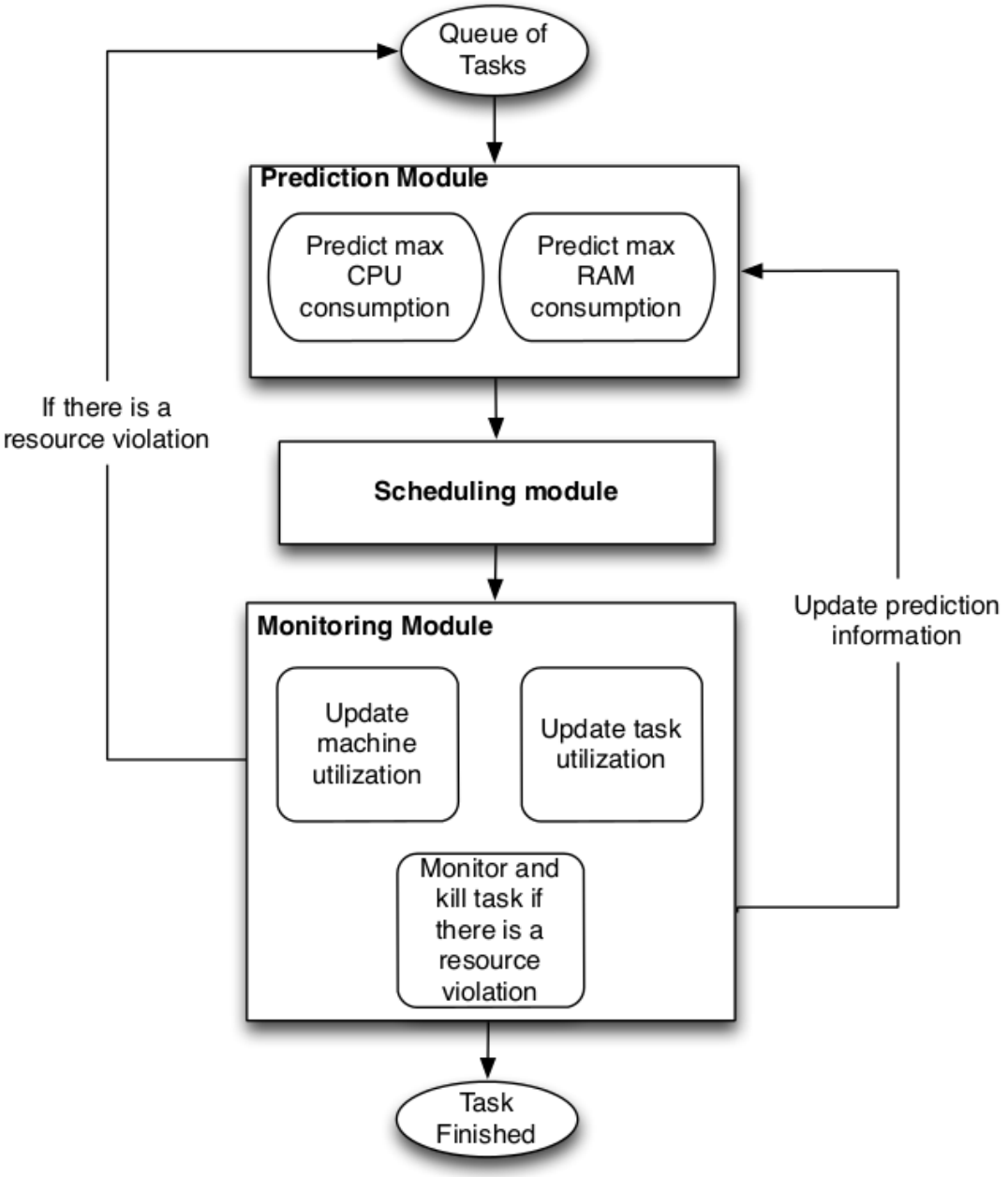


Figure 5.1: Overview of the proposed approach structured around three modules : Prediction, Scheduling and Monitoring.

used as features for the learning techniques and output variables for this module:

5.2.1 Input

Extracted from the Google dataset presented in Section 2.3.5, the input attributes used for predicting resource requirements of tasks (i.e., explanatory variables) are:

1. *CPU limit*: The maximum amount of CPU that a task is expected to use as specified by a user.
2. *RAM limit*: The maximum RAM a task is expected to use as specified by a user.
3. *Number of tasks*: The total number of tasks that belong to a job.
4. *Priority*: The priority of a task, represented by an integer. The trace uses twelve task priorities (numbered 0 to 11). Production tasks have *High* priority (i.e., 9 - 11), tasks of *Normal* importance have a priority ranging from 2 to 8, and *Free* tasks have a priority of 0 or 1 [RTG+12]. High priority tasks are processed before tasks with low priority. When submitting tasks to the cloud system, end-users are invited to pay a fee to increase to priority level of their tasks.
5. *Scheduling class*: The latency sensitivity of a task, represented by a small integer, ranging from 0 to 3, where 3 represents a more latency sensitive task (e.g., serving revenue-generating user requests), and 0 representing non-production tasks [RWH11].
6. *User*: The name of the user that submitted the job, represented by an obfuscated string [RWH12]. As the prediction is limited to numbers, this string is transformed to a natural number, where each number represents a unique user.
7. *Logical job name*: The logical name of the job (i.e., name of the application) submitted, represented by an opaque base64-encoded string. The trace documentation states that different executions of the same program will usually have the same logical name. Similar to the user's name, this string is transformed to a unique natural number.
8. *Job name*: The name of the job submitted, represented by an opaque base64-encoded string. This attribute is sometimes generated by automated systems to avoid conflicts (e.g., for MapReduce). In the same fashion as the previous two variables, the string is transformed to a natural number.
9. *Different machine constraint*: A binary attribute that indicates whether a task

from a job must be scheduled to execute on a distinct machine than any other running task in the job [RWH11].

For completeness, one should note that the non-numeric variables were transformed to integers, as they could not be used as input for the predictors in their original format. As an alternative way to proceed, one could use the *one-hot-encoding* to perform such a transformation.

5.2.2 Output

The output variables of this module are $predCPU$ and $predRAM$ which can be thought as an alternative expected peak consumption to the one provided by the user. We claim that well-suited machine learning techniques can provide more accurate estimates than the user would. For these predictions, two models are defined, one for each resource. Both models receive the same input attributes for the prediction. We describe below the three prediction techniques used in this work:

$UL_{\%}$ **(User Limit Percentage)**

This ad-hoc prediction technique learns which fraction of the user's stated resources limit yields the lowest root mean square error (RMSE) against the actual peak resource consumption. We motivate the use of such a predictor by referring back to Figure 2.6 showing that there is a general tendency from users to overestimate peak consumption. In order to produce more accurate estimates, we thus seek to learn and cancel the average peak resource overestimating behaviour.

Finding the values $UL_{\%}$ that minimises the RMSE against the actual peak is done as follows. We considered a set of tasks used for training and incrementally decreased by 1%, from 99% to 1%, the user stated resources limit. The portion of the user stated limits that minimises the RMSE against the tasks actual peak learned by this predictor will be used to produce peak resource estimation for unseen incoming tasks. For example, if the minimum RMSE is achieved with 40% of the user stated CPU limit, the $UL_{\%}$ model will use 40% CPU limit for the next incoming tasks.

Multiple Linear Regression

This is an extension of simple linear regression, which is a standard approach that fits a line that minimises the sum of the squared residuals. Multiple linear regression allows us to predict the value of a response variable (i.e., $predCPU$ or $predRAM$) based on the value of the above mentioned explanatory variables.

The goal of multiple linear regression is to model the relationship between the explanatory and the response variables.

According to [Wei14], the values of the explanatory variables x are associated with a value of the response variable y. The regression line for p explanatory variables $x_1, x_2, ..., x_p$ is defined to be $\mu_y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p$. This line describes how the mean response, μ_y, changes with the explanatory variables. The observed values for y, range about their means μ_y, and are assumed to have the same standard deviation, σ. The fitted values estimate the parameters $\beta_0, \beta_1, ..., \beta_p$ of the regression line. We can interpret β_i as the average effect on y of a one unit increase in x_i, holding all other predictors fixed.

The multiple linear regression model is expressed as **data = fit + residual**, where the **fit** term represents the expression $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p$. Furthermore, the **residual** term represents the deviations of the observed values y from their means μ_y. The notation for the model deviations is ϵ.

Formally, the model for multiple linear regression, given n observations, is:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} + \epsilon_i \quad for\ i = 1, 2, ...n$$

In multiple linear regression each variable can then be calculated independently. The most efficient way of doing so involves linear algebra. In practice, this can be easily calculated in R using the lm function, which is part of the $stats$ package.[1] The lm function predicts the output using the Covariance and Standard Deviation of the underlying data, which can be computed easily.

Random Forest

This is an ensemble learning method that uses trees as building blocks to construct more powerful prediction models, namely many decision trees are constructed during the training phase, such that the collection of trees (*forest*) selects the average over all the trees. The method merges the concept of bagging and random selection.

According to [HTF09], bagging or bootstrap aggregation is a technique for reducing the variance of an estimated prediction function. Given a standard training set D of size n, bagging generates m new training sets D_i, each of size n', by sampling from D uniformly and with replacement. By sampling with replacement, some observations may be repeated in each D_i. Bagging works

[1] https://stat.ethz.ch/R-manual/R-devel/library/stats/html/lm.html

in particular for high-variance (i.e., high error from sensitivity to small fluctuations in the training set), low-bias (i.e., low error from erroneous assumptions in the learning algorithm) procedures, such as trees. As in bagging, random forest builds decision trees on bootstrapped training samples. But when building these decision trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors. A fresh sample of m predictors is taken at each split, and the number of predictors considered at each split is equal to the square root of the total number of predictors [SMT09] [JWHT13].

The rationale behind selecting only a small subset of the predictors is that in many cases there is one strong predictor in the data set, along with a number of other moderately strong predictors. Therefore, in the collection of bagged trees, most or all of the trees will use this strong predictor in the top split. Consequently, all of the bagged trees will look highly correlated. Random selection of variables overcomes this problem by forcing each split to consider only a subset of the predictors. Therefore, on average $(p - m)/p$ of the splits will not even consider the strong predictor, and thus other predictors will have more of a chance. This process is called *tree decorrelation*, thereby making the average of the resulting trees less variable and hence more reliable [JWHT13].

Another interesting characteristic of random forest is that they do not over fit due to the law of large numbers [Bre01]. Furthermore, we implemented this predictor on the R programming language, using the $randomForest$ function, which is part of a package with the same name.

5.3 Scheduling Module

This module schedules tasks to machines while trying to satisfy a set of constraints subject to a multi-criteria objective function. Below we describe some notation followed by the optimization model of the problem that is solved at a given time-point. This model is focused on the description of the cloud system provided in [RWH11]. This model is therefore tailored to capture operational constraints relevant to this particular cloud setting.

5.3.1 Mathematical Model

Notation

- Let $\mathcal{J}$ be the set of jobs that are currently in the queue. This set is composed of recently arrived tasks and those that are recently evicted from their previously assigned machines. A job is represented by a set of tasks.
- Let $\mathcal{T}$ be the set of all tasks associated with all jobs that need to be scheduled.
- Let $\mathcal{O}$ be the set of all tasks that are already running on the machines. Let o_i be the machine on which task $i \in \mathcal{O}$ is currently running. Let p_i be the time when task $i \in \mathcal{O}$ started on machine o_i.
- Each task is associated with a priority and a scheduling class that denotes how important and latency sensitive a task is respectively. Let S be the set of priorities. Let $s_i \in S$ be the priority of task i and let w_{s_i} denote a non-zero positive integer constant associated with the priority s_i. Let C the set of scheduling classes. Let $c_i \in C$ be the scheduling class of task i and let w_{c_i} denote a non-zero positive integer constant associated with the scheduling class c_i. The higher values of these constants mean that they are more important and latency sensitive respectively.
- Let a_i be the arrival time of task i. Let d_i be a non-negative integer constant that denotes the duration for each task $i \in \mathcal{T} \cup \mathcal{O}$. Let e be the current time. Let $l = e + \sum_{i \in \mathcal{T}} d_i + \max\{(p_i + d_i) \mid i \in \mathcal{O}\}$ be a constant that denotes the latest finish time of any task. Let $H = \{e, e+1, \ldots, l\}$ be the current horizon during which tasks should be scheduled.
- Let M be the set of all machines. Let M_i be the set of machines on which task $i \in \mathcal{T}$ can be scheduled.
- Let R be the set of resources. In this chapter we only consider CPU and RAM. Let C_{mr} be the capacity of machine $m \in M$ for resource $r \in R$.
- Let q_{ir} be the predicted value for resource $r \in R$ required by task $i \in \mathcal{T} \cup \mathcal{O}$. Notice that different ways of prediction will have different impact on the actual utilisation of servers.
- Let $F \subseteq \mathcal{J}$ be a set of jobs whose tasks must run on different machines.

Variables. Let x_i be an integer variable that denotes the machine on which task $i \in \mathcal{T}$ is running. The domain of the variable x_i is M_i. Let y_i be an integer variable that

denotes the time at which task i starts. The domain of y_i is H. Let u_{mrh} be a continuous auxiliary variable that denotes the usage of machine m for resource $r \in R$ at time $h \in H$.

Constraints We modeled the following operational constraints:

Assignment Constraint. Each task $i \in \mathcal{T}$ must be assigned and processed to completion on a machine. This is trivially satisfied by selecting a value from M_i and assigning it to x_i. Each task is also assigned the time at which it starts executing on the machine x_i by assigning a value to y_i from the set H.

Capacity Constraint. The resource utilisation u_{mrh} of a machine m cannot exceed its capacity on resource r at any time h:

$$\forall_{m \in M} \forall_{r \in R} \forall_{h \in H} : \quad u_{mrh} = \sum_{\substack{\forall i \in \mathcal{T}, x_i = m \\ y_i \leq h < y_i + d_i}} q_{ir} + \sum_{\substack{\forall i \in \mathcal{O}, o_i = m \\ p_i \leq h < p_i + d_i}} q_{ir}$$

$$\forall_{m \in M} \forall_{r \in R} \forall_{h \in H} : \quad u_{mrh} \leq C_{mr}.$$

Conflict Constraint. The tasks associated with any job in F must be scheduled on different machines:

$$\forall_{f \in F} : |\{x_i | i \in f\}| = |f|$$

.

Objective Function The objective is to minimise the total waiting time. The experimental section will also analyse the impact of the consolidation techniques on the average machine utilisation.

Waiting Time (WT). Total sum of the tasks' weighted WT:

$$WT = \sum_{i \in \mathcal{T}} w_{s_i} \times w_{c_i} \times (y_i - a_i)$$

.

As mentioned in Section 5.2.1 the scheduling class (WSI) is a measure of how latency-sensitive a task is. On the other hand, the priority (WCI) of the task distin-

guishes between *free* and *production* tasks. Our measure of the weighted waiting task has therefore to take into account these two aspects. The coefficients are multiplied amongst them and then multiplied with the actual waiting time of the task. Minimizing this quantity allow us to favor tasks with high priority and scheduling class to be scheduled with minimal waiting time.

Machine Utilisation (MU). Let $E = \{e, l\} \cup \{y_i, y_i + d_i | i \in \mathcal{T}\} \cup \{p_i, p_i + d_i | p_i > e, i \in \mathcal{O}\}$ be a set of time-points when the utilisation profile of any machine can change. Let $\langle h_1, h_2, \ldots, h_n \rangle$ be an ordered set of time-points such that $h_k < h_{k+1}$, $h_k \in E$, $h_{k+1} \in E$ and $n = |E|$. Let $E^* = E - \{h_n\}$. The cumulative utilisation of a machine m for resource r is denoted by CU_{mr} and it is computed as follows:

$$CU_{mr} = \sum_{k \in \{1..|E^*|\}} \begin{cases} (u_{mrk}/C_{mr}) \cdot (h_{k+1} - h_k) & \text{if } u_{mrk} > 0 \\ (h_{k+1} - h_k) & \text{otherwise.} \end{cases}$$

The average utilisation of a machine for a given resource r of a given cluster is:

$$MU_r = \frac{\sum_{m \in M} CU_{mr}/(C_{mr} \cdot (h_n - h_1))}{|M|}.$$

5.3.2 Policies for Online Scheduling

We are considering a complex online scheduling problem for which we have incomplete information at decision time. The incompleteness of the information is due to both the online nature of the problem and the fact that the task resource requirements are uncertain. This section discusses possible approaches to building policies to solve the problem at hand. Naturally any scheduling policy to be used in this setting should be able to reach a decision in a very limited amount of time. In order to guarantee a good level of reactiveness, fast scheduling techniques based on a complete ordering of the machines can be used. Amongst these simple policies *Random* and *Round Robin* scheduling policies can be implemented but are expected to perform rather badly in terms of overall machine utilisation. The reason for that is that these policies naturally spread the workload over the entire cluster of machines, which in turn leads to machines running far from their maximum capacity.

As an algorithmic enhancement to these simple policies, we implement a greedy heuristic that tries to spread the workload over a minimal subset of available machines. This concept is illustrated in Algorithm 7. The algorithm maintains two sets of ma-

chines: the set of machines that are *ON* is denoted by M_o. These machines are currently running tasks and should be candidates for incoming tasks. On the other hand, the set of machines that are in *standby* mode is denoted by M_s. Any machine in M_s is not currently processing tasks and thus could be switched off.

Algorithm 7: *schedule*

Input: $p \in \mathcal{T}$, M_o, M_s
Output: m, t

```
1  minw ← 0
2  maxw ← 2
3  Loop
4      ⟨m, t⟩ ← findMachineTime(p, M_o, minw, maxw)
5      if m == Null then
6          ⟨m, t⟩ ← findMachineTime(p, M_s, minw, maxw)
7          if m ≠ Null then
8              M_s ← M_s \ {m}
9              M_o ← M_o ∪ {m}
10     if m == Null then
11         minw ← maxw
12         maxw ← maxw × 2
13     else
14         return ⟨m, t⟩
```

The first idea leveraged here is to find a feasible packing of incoming tasks within M_o. One could, in fact reduce the set M_o to a single machine and simply extend the packing as far in time as one needs. This rather extreme case could achieve optimality in terms of machine utilisation but it would perform very poorly in terms of waiting time by delaying the execution of tasks by hours. On the other hand, using all the machines available in the cluster at any point in time would reduce the waiting time to a non-significant value while decreasing the utilisation of individual machines significantly. Our greedy methods implement a way to find a balance between the two extreme scenarios mentioned above.

To minimise the impact on the average waiting time, we define the initial bounds denoted by `minw` and `maxw` (Lines 1 and 2). For the experiments reported in this chapter the initial upper bound on the waiting time is set to a value of 2.0 seconds.[2] The procedure is a repeat loop that breaks when a suitable machine and starting time was found for the task under consideration. The algorithm first tries to find a machine within the

[2]The mean task waiting time in the trace is approximatively 2 seconds. Setting the upper bound to 2 seconds guarantees that the waiting time in the schedule resulting from our optimisation is at most the one in the original trace.

Algorithm 8: $findMachineTime$

Input: $p \in \mathcal{T}$, M, $minw$, $maxw$
Output: m, t

```
1 s ← 0
2 t ← 2
3 while |M| ≠ ∅ do
4     m ← pop(M)
5     t ← consistent(p, m, minw, maxw)
6     if t ≠ Null then
7         return ⟨m, t⟩
8 return ⟨Null, Null⟩
```

set of active machines subject to the bounds on the waiting time (Line 4). If no such machine is found then it tries to find a machine that is in *standby* mode, and update the sets M_o and M_s respectively (Line 5–9). If the machine is still not found then the bounds on waiting time are relaxed (lines 11 and 12) and the above steps are repeated. This relaxation guarantees that the algorithm terminates since there will be a valid assignment by delaying the starting time of the task under consideration. As soon as a valid assignment is found, the selected machine and the time at which the task is scheduled is returned (Line 14).

The pool of machines is thus managed as follows: the scheduler might bring a new machine formerly in *standby* mode into the subset of busy machines. On the other hand, the monitoring module might put an *idle* machine in *standby* mode if no task is running on it. Algorithm 8 iteratively selects and removes a machine m from the set M. The selection operation is implemented as either *Random* or *Round Robin*. It invokes $consistent$ to check if the task p can be scheduled on the selected machine m within the allowed bounds of waiting time and if it has enough CPU and RAM capacity to accommodate the task and if it is not currently processing another task coming from the same job if the job needs to be run on different machines. If the machine is suitable for the task under examination, the algorithm returns both a pointer to that machine and the earliest starting time. If the set M becomes empty the algorithm returns without any machine or time, leaving Algorithm 7 the task of either bringing a new machine into the set of busy machines or further relaxing the upper bound `maxw`.

Algorithm 9: $checkForEviction$

Input: m
Output: K

1 $K \leftarrow [\,]$
2 $candidates \leftarrow sort(\{i \in \mathcal{O} \quad | \quad o_i = m\}, \quad order)$
3 **while** $overloaded(m)$ **do**
4 $\quad K \leftarrow K \cup pop(candidates)$
5 return K

5.4 Monitoring Module

The primary function of the monitoring module is to observe and record the actual resource requirements of each task periodically which can be used by both prediction and scheduling modules. The prediction module can use this recorded data to improve the prediction function further. The scheduling module can benefit by updating its resource utilisation profile in two cases: (i) if the maximum resource consumed by a task is higher than the amount given by predictor module, then the maximum resource requirement of the task can be updated and hence the utilisation profile; and (ii) if the task is finished, then the utilisation profile of the machine used by the scheduling module can also be updated in the cluster.

Another function of this module is to decide the eviction of some task(s) if the resource requirements of the tasks running on the machine is higher than its maximum capacity. The selected tasks would be marked for eviction, their execution stopped and sent back to the *Queue of Tasks* in order to reduce the level of utilisation of the machine. Note that any eviction policy has to reach a decision with a knowledge local to each machine. There are indeed several strategies that one could implement to select candidate tasks for eviction. The basic strategy, implemented by the providers of the trace consists in choosing tasks for eviction based on their priority level. The reason for this is that the system must guarantee a high quality of service for tasks with high priorities. We explore two alternative implementations of the eviction policy that simply relies on different orderings of the candidate tasks. Note that these eviction policies are triggered by resource overload on the machines. It is thus possible for a task to be repeatedly evicted until the machine frees up some resources. Algorithm 9 thus details the implementation of line 18 of the simulation Algorithm 10.

We study the following three orderings that could be used on Line 2:

minPrio

This is the policy originally used by the providers of the trace. Tasks are ordered

by increasing priority. Tasks with smaller priorities are favored for eviction.

minRunningTime

The tasks are ranked by increasing running time. Tasks that have been running for the least amount of time are favored. The underlying idea is that one should avoid evicting tasks that have been running for a long period of time. Eviction tasks that have been running for long periods of time would impact negatively the average waiting time.

minNumTask

In this case the candidates are ordered by decreasing actual resource requirements. The idea here is to try to favor the eviction of tasks requiring a large amount of resource, so that we locally minimise the number of evictions.

Algorithm 9 simply sorts the currently running tasks on the machine and greedily mark for eviction tasks until the machine's capacity is not exceeded anymore. All these strategies will be compared in Section 5.5.

To conduct experiments in this complex scheduling environment, we developed an event-driven simulation framework. This framework handles and maintains a collection of ordered events related to tasks along a time line. At any point in the simulation, the collection of events is carefully handled in such a way that it remains ordered by increasing time stamp. The simulation framework that is used to carry experiments is described in pseudo-code in Algorithm 10. The simulation is bootstrapped on Line 1 by initializing the queue of events E with the first task arrival selected from the set of arrival events. We next define the set of events dynamically generated as the simulation unfolds according to Algorithm 10:

- `Arrival.` This event simulates a task submitted for scheduling. Upon arrival, the scheduler is called to assign the task to a machine and decide on the starting time (Line 5). This step also includes a call to the prediction module that returns an estimate on the peak values that the task is expected to reach. The scheduler guarantees to find a pair $\langle x_t, y_t \rangle$ with x_t expressing to which machine the task has been assigned and y_t the starting time for the task to be processed on that machine. We then add, on Line 6, the `Start` event associated with the current task at time y_t on the simulated time line; meaning that task t is due to start when the simulation reads that event.

  On Lines 7 to 9 we feed to the simulated time line E with as many - evenly spaced in time - `resource Update` as we could retrieve from the original data contained in Google trace. These events are capturing the actual resource

consumption of the task at hand. It is recalled that d_t is the duration of the task t and k_t is the number of records of resource usages captured by Google traces.

Lastly, on Line 10 we place the `Arrival` event associated to the next task coming into the simulation. From a data loading point of view, this is done lazily by only adding a task arrival to the queue when the current one has just been handled.

- `Start`. This event simulates the task's execution starting on the machine to which it has been assigned(Line 12). From this point and on, the task is running on the machine consuming some resources and thus it is eligible for eviction if the machine would overload. Moreover, since the task duration is considered known in our setting, we simply add, on Line 13, a `Finish` event further down the simulated time line E at time $y_t + d_t$.

- `Finish`. Once the simulation reaches a `Finish` event, the associated task is considered completed and thus removed from the machine freeing any resource that the task was consuming in the process. This is implemented on Line 15.

- `Update`. The update event, on Line 17, simulates the variation on the task's requirements on both CPU and RAM resources. As stated previously, in most cases, the actual requirements are less than the amount of resource that were provisioned for the task. Nonetheless, it might happen that the actual consumption exceeds the provisioned space, as checked on line 18. In that case, the machine running the task might find itself in a saturated state, which triggers an eviction of the task itself or, any other task running concurrently on the machine. In addition, since evictions are triggered when a task requires more resources than forecasted, we update the predicted peak required when it was underestimated. In that way, we continuously learn about peak requirements of tasks.
  For each tasks marked for eviction, we generate, on Line 20, an `Evict` event happening an arbitrarily small delay in milliseconds α further on the simulated time-line. Several eviction policies are discussed in Section 5.4.

- `Evict`. This event simply removes any other events related to the evicted task, on Line 23 and emits a new `Arrival` event on Line 22. This new `Arrival` event is set to happen an arbitrarily small (α) number of ms later in the simulation.

Using the definition of these events, we build a simulated time line of events occurring in a logical order imposed by the time of occurrence of an event.

Algorithm 10: $simCluster$

```
1  E ← ⟨t_0, a_t, arrival⟩)
2  while E ≠ ∅ do
3      Pop next event e = ⟨t, time, etype⟩ from E
4      if etype == arrival then
5          ⟨x_t, y_t⟩ ← predictAndSchedule(t, time)
6          Insert ⟨t, y_t, start⟩ in E
7          w ← d_t/k_t
8          for i = 0, i < k_t, i++ do
9              Insert ⟨t, y_t + w * i, update⟩ in E
10         Insert ⟨t + 1, a_{t+1}, arrival⟩ in E
11     else if etype == start then
12         Add t to the list of running task on x_t
13         Insert ⟨t, y_t + d_t, finish⟩ in E
14     else if etype == finish then
15         Remove t from x_t and update resource usage
16     else if etype == update then
17         Update resource consumption of t on x_t
18         K = checkForEvictions(x_t)
19         for t' ∈ K do
20             Insert ⟨t', time + α, evict⟩ in E
21     else if etype == evict then
22         Remove all the events related to t from E
23         Insert ⟨t, time + α, arrival⟩ in E
```

5.5 Experiments

In this chapter, we utilised the first 24 hours of the trace for creating the initial prediction models. Then, we used the next 24 hours for testing the accuracy of our methodology. We did not use the entire 29 day trace due to the complexity of the machine learning techniques together with the complexity of analyzing such large amounts of data collected in the 29 days. Moreover, previous work shows that the characteristics of the trace remained similar within the 29 days of the trace [RTG+12]. Thus, we expect similar results when extending the experiments for the whole duration of the trace. Finally, we only considered tasks that started and finished within the 48 hours studied.

5.5.1 Experimental Setup and Error Metrics

In order to perform experiments we simulated the trace provided by Google for which we developed the event-based simulation as described above. The implementation of the simulation was done using Python 2.7. Experiments were conducted on an 8-core Intel Xeon E5-2640 Processor (2.50GHz), limited to 8GB RAM.

A task is added to the queue based on its arrival time or when it is evicted from the machine. A task is removed from the queue as soon as the scheduler allocates it to a machine. Each task is associated with the actual current values of resource requirements which are updated periodically based on the values provided in the trace over its duration.

In order to predict resource consumption optimally, it is necessary to update the prediction model periodically. As such, we updated the prediction model after each hour of the simulation, for both CPU and RAM, with the newly measured maximum resource utilisation of the tasks that finished within that hour, together with their attributes. This is implemented in order to increase the accuracy of the prediction model over time. The execution time used by the predictor to update its model was not considered. Furthermore, we present the results of the scheduler for each 4 hours cumulative time window.

We computed the significance of the variables (i.e., p-value) of the input variables for constructing the multiple linear regression model. As all variables have a p-value lower than 0.005, we used all of them for constructing the model. We also noticed that the predictor outcome was a negative value for some tasks. We replaced those values by a new minimum value, which was explained in Section 2.3.5 (i.e., 6.247e-7 in the case of CPU, and 9.53e-7 in the case of RAM). Furthermore, in the case of the random forest predictor, the number of trees used in the model was set to 100, since we noticed that increasing the number of trees after this amount did not bring any benefit in terms of accuracy, but only an increment in the algorithm's execution time. Moreover, the number of variables randomly sampled as candidates at each split was set to 3. This was defined by following a rule of thumb that states that the number of variables randomly sampled should be equal to the square root of the number of input attributes, which in our case is 9 [SMT09].

Metrics In order to measure the accuracy of our prediction model, we use two error metrics: the mean absolute error (MAE) and the root mean squared error (RMSE). The first measures the average of the absolute errors, where the errors are calculated as the

difference between the predicted and the actual value. The latter measures the standard deviation of the absolute errors. Thus, it gives more weight to large but infrequent errors [CD14].

Furthermore, we consider the following metrics for measuring the performance of our scheduler: (i) the number of active machines, (ii) the cumulative average utilisation of the entire cluster, (iii) the number of evictions for each of the predictors, and (iv) the average task waiting time. We expect the number of active machines to decrease, while the utilisation level of the entire cluster to increase, without severely affecting the task waiting time.

5.5.2 Predicting CPU and RAM Maximum Utilisation

Figure 5.2 and Figure 5.3 shows the MAE and RMSE values for the prediction models and the user defined limit for the maximum CPU and RAM utilisation. We observe that all the predictors generate fewer errors for the CPU than the user's limit. We observe small absolute differences between the user's limit and the other prediction techniques. However, as the values are normalized in the Google trace, a small absolute difference can represent a large improvement with regards to the actual resource value. Therefore, we are interested to study whether these can have a significant impact on the schedulers' performance.

The figure shows how simple predictors can produce an average reduction in MAE by 39.5% in the case of $UL_{\%}$, and 53.5% when using multiple linear regression (Multiple LR). Moreover, random forest outperforms the user's limit in average by 79.8% in terms of MAE and by 66.1% with regards to RMSE. However, random forest is a very computational intensive predictor, that requires a significant amount of time for the creation of its model.

Nevertheless, we are aware that optimizations through parallelisations could decrease the execution time significantly. Moreover, we observe in Figure 5.3 that the user's estimate on the RAM requirement is more accurate than its estimate for CPU. However, the reduction in error between the three predictors and the user's limit is higher for RAM estimates. The average MAE reduction between the user's limit and $UL_{\%}$ is 68%. In the case of Multiple LR, the average error reduction is by 71.6% for MAE, and 67.3% for RMSE. Similar to CPU predictors, we notice that random forest produces the lowest error. Random forest achieves a reduction by 89.6% for MAE and 80.3% for RMSE, compared to the user's limit.

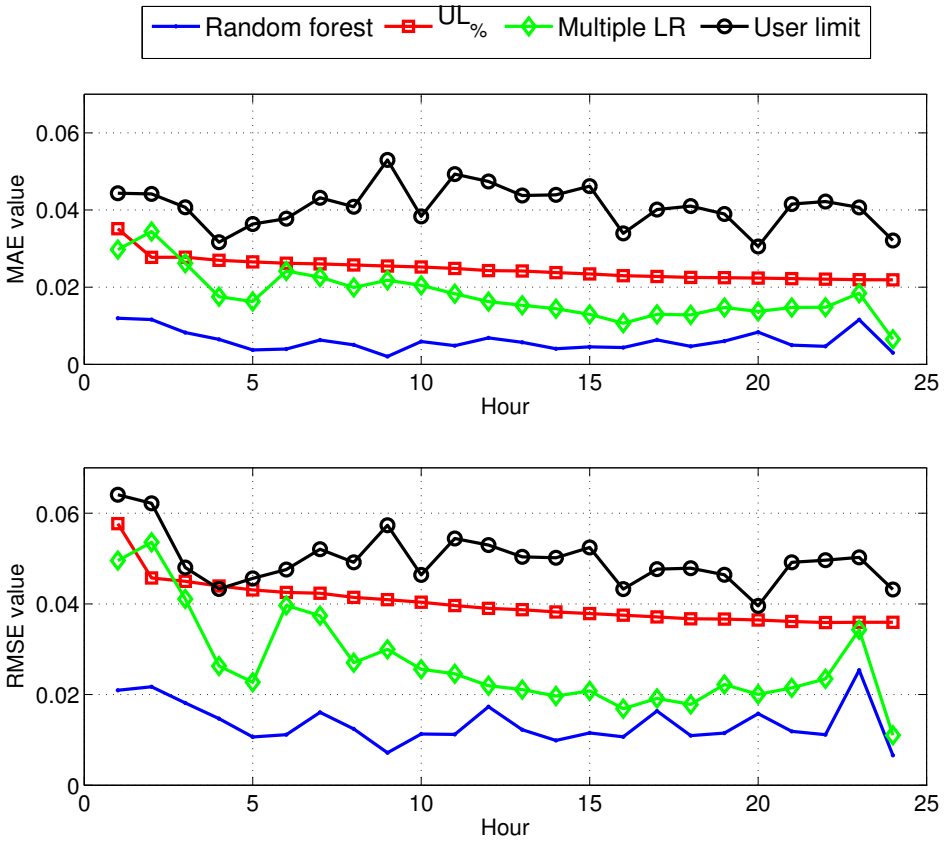


Figure 5.2: MAE and RMSE values for all the CPU predictors and user defined limit

We present below the impact of these reductions in the estimations errors and how they bring large benefits in terms of number of active machines and their utilisation levels.

5.5.3 Evaluating the Scheduling Policies

In this section, we explore the performance of the scheduler using the following policies: Round-Robin, and Random with and without the greedy algorithm detailed in Section 5.3.2. Under the hypothesis of perfect information about the peak requirements of tasks, we show that Algorithm 7 improves significantly the performance of simple heuristics such as *random* assignment and *round robin* assignments. We then drop the perfect information assumption and analyse the interactions between the scheduler and the various predictors. Finally, the eviction policies are brought into the picture and discussed.

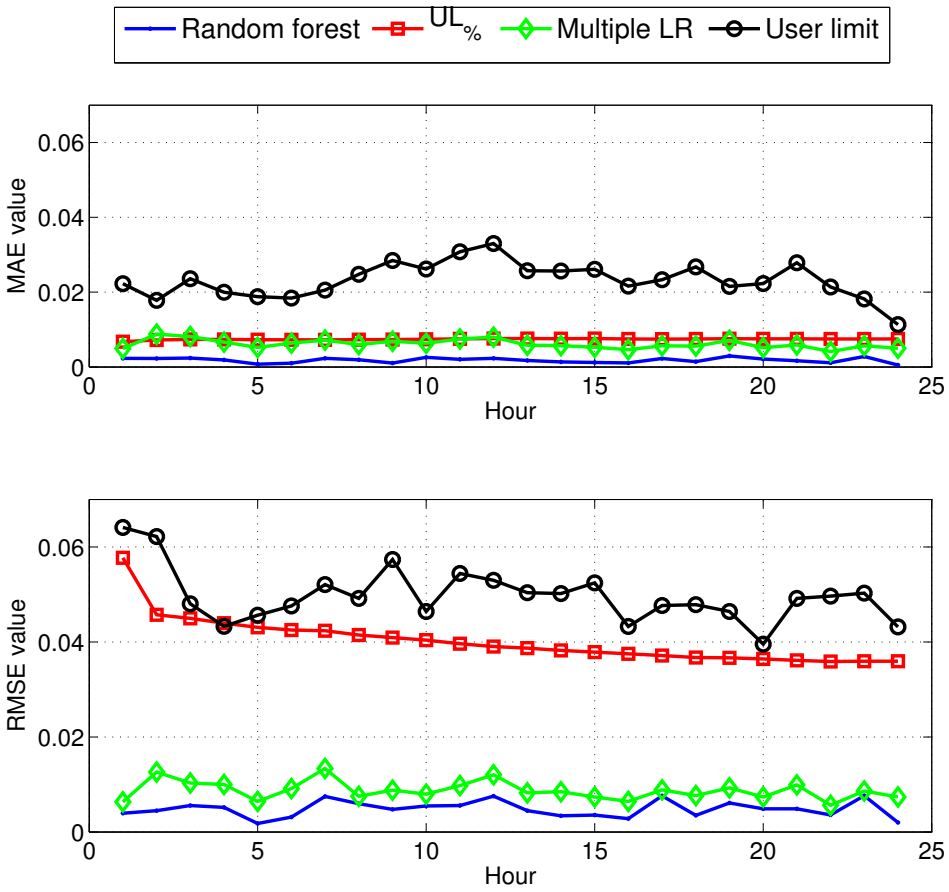


Figure 5.3: MAE and RMSE values for all the RAM predictors and user defined limit

5.5.3.1 Polices for Known Peak Resource Requirements

The greedy principle implemented by Algorithm 7 is evaluated in Figure 5.4. This figure shows the number of machines used over 24 hours to address the workload, by the *random* and *round robin* policies and the same polices applied on a restricted set of machines using Algorithm 7. As expected, both policies, perform rather poorly in terms of machines used to process the tasks. This is due to the fact that both policies naturally spread the tasks over all the machines without trying to somehow optimize the utilisation of single machines. On the other hand, by applying the same polices on a restricted set of machines as discussed in Section 5.3.2 the plot clearly shows the advantage of trying to reuse machines already processing tasks. In fact, in the case of the *round robin* policy, the mean number of machines used at the same time was $7,935.7$ with a maximum of $12,398$ against a mean of $2,009.8$ and maximum of $4,658$ for the restricted counterpart.

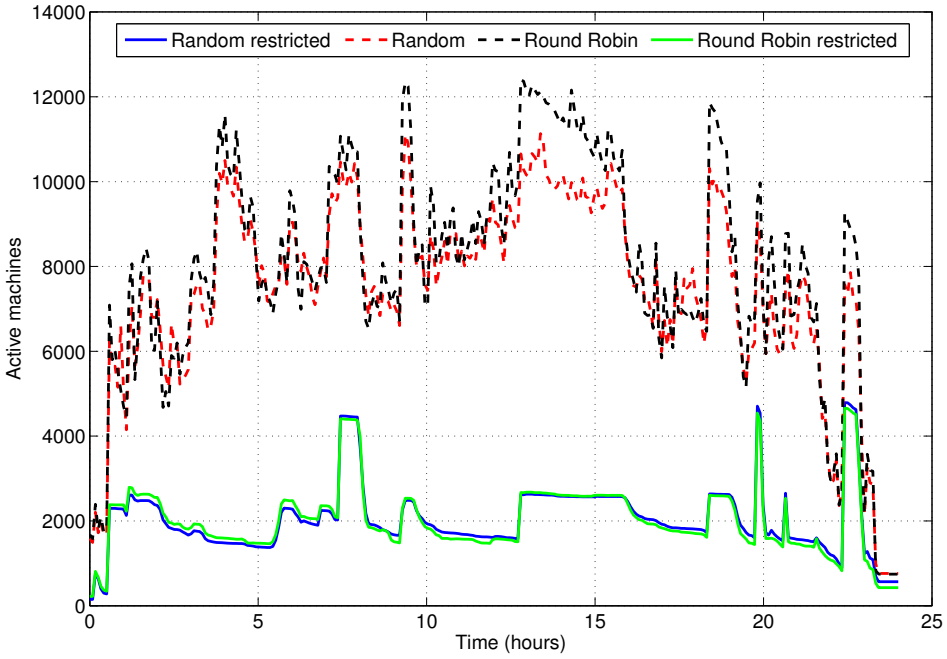


Figure 5.4: Number of machines used by Random and Round Robin polices on the with and without using the greedy scheme.

Table 5.1: Machine usage statistic for Random, Round Robin policies and their restricted counterparts under perfect information from the predictors.

	Random	**Random restricted**	**Round Robin**	**Round Robin restricted**
Avg used machines	7459.2	**2024.9**	7935.7	**2009.8**
Max used machines	11133	**4789**	12398	**4658**
Evictions	0	0	0	0
Average WT	0.0	0.0008	0.0	0.0008
Max WT	0.0	1.99	0.0	1.99
CPU Util	41.00%	**50.75%**	44.10%	**61.04%**
RAM Util	41.43%	**49.96%**	47.42%	**61.15%**

Since we are assuming an oracle providing us with perfect knowledge of the peak requirements (i.e. tasks' actual behaviour), the eviction mechanism is never triggered for all the aforementioned polices. Furthermore, as can be seen in Table 5.1, there seems to be a trade-off between CPU and RAM utilisation and the average waiting time. Using our greedy algorithm, we allow tasks to be scheduled within the 2 seconds window which allows to assign more tasks to a single server. In turn the CPU and RAM utilisation are much better since we are allowing to tighten the schedules locally on machines.

5.5.3.2 Policy-Predictor Interactions

We now drop the assumption that we can provide the scheduler with perfect prediction of the tasks' peak requirements which makes the analysis more difficult but much more realistic. We discuss the impact of the various predictors on the scheduling decisions. Figure 5.5 presents the total number of active machines during the 24 hours of our simulation. The number of active machines is updated every 5 minutes. A machine is considered active if its CPU or RAM utilisation is higher than 0%. The figure shows the results for two of the proposed prediction techniques. Results from the ad-hoc prediction technique, namely $UL_{\%}$, are not presented due to limitations of our tool when handling huge numbers of evictions. We ran $UL_{\%}$ for 9 hours and noticed almost the same number of evictions than when running Random forest for 24 hours. Large number of evictions directly impact the task waiting time, number of active machines, and the levels of utilisation of the machines, therefore the $UL_{\%}$ predictor would undoubtably have produced the worst results. Moreover, the figure also presents results for a perfect predictor, namely *Actual Peak*, and the user limit. The *Actual Peak* represents a theoretical scenario, where the predictor knows the actual maximum resource consumption for all the incoming tasks. On the other hand, one can think about the user limit as a practical baseline since it is the only information accessible at scheduling time in the original description of the problem.

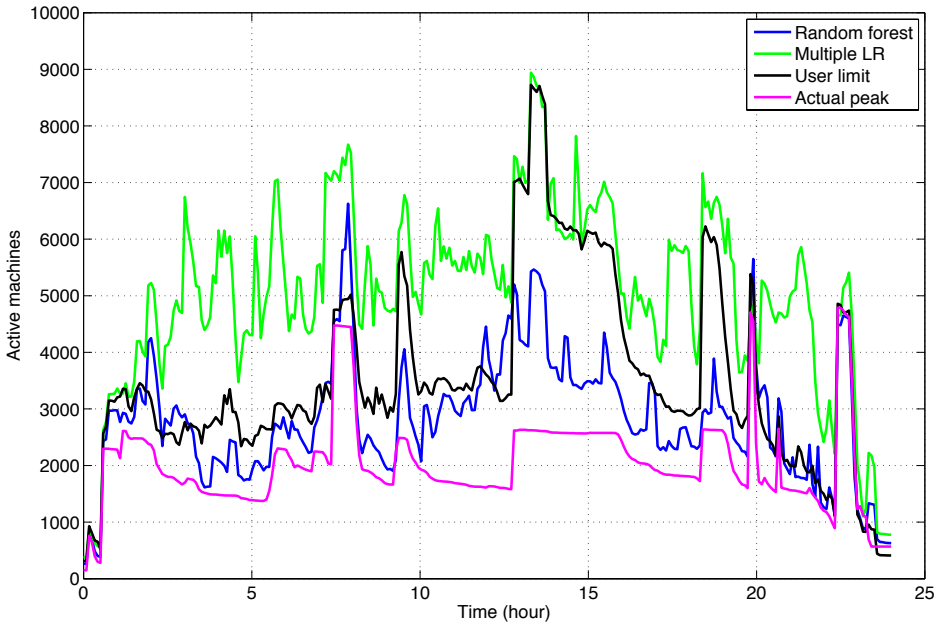


Figure 5.5: Active machines (considering windows of 5 minutes)

As expected, *Actual Peak* achieves the lowest number of active machines at almost any

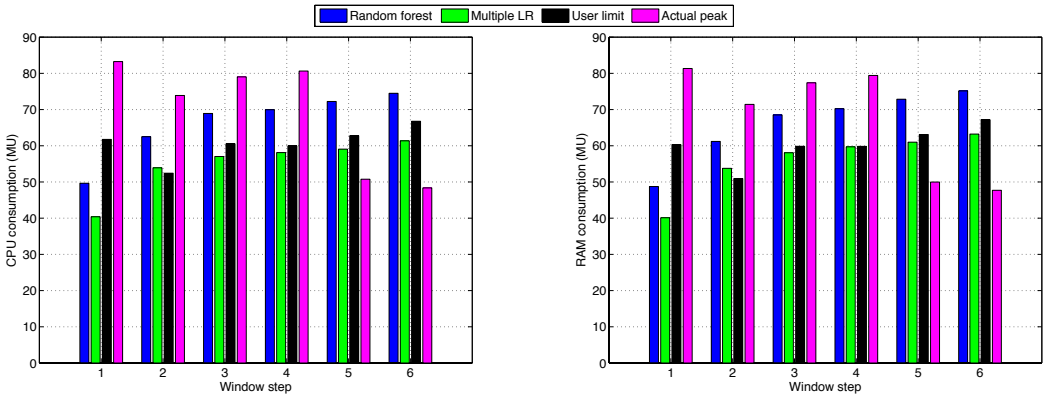


Figure 5.6: Cumulative average CPU (left) and RAM (right) utilisation

given time. We observe that Random forest is the second predictor in terms of lowest number of active machines. In comparison to UL, random forest uses on average 671.9 fewer machines, with a maximum difference of 3,337 fewer machines. Moreover, in 18.9 hours out of the 24 hours studied, Random forest used fewer machines than UL. The figure also shows that Random forest and Multiple LR have a more consistent number of active machines than UL. For instance, we noticed two times periods in which both predictors significantly outperform UL, namely the range between 12.83 and 16.75 hours, and between 18.41 and 19.08 hours, in those two periods the average difference in terms of active machines between Random forest and UL is more than 2,300 machines.

Nevertheless, in contrast to their results in MAE and RMSE, we notice that the other two predictors, $UL_{\%}$ and Multiple LR, are in most cases outperformed by UL. We observe that UL uses on average 1,511 fewer machines than Multiple LR. These results suggests that MAE and RMSE are not enough to assess the impact of the predictors when used for scheduling purposes. The high number of machines utilized by Multiple LR is caused by generally under estimating the actual peak of a task, which causes the number of evictions and thus the number of machines to increase. However, the time for which these large machines are active is in many cases only a few minutes, since the majority of the tasks that are evicted are tasks with short duration.

We present in Figure 5.6 the cumulative average utilisation of every 4 hours for all the machines in the cluster. We measure the CPU and RAM consumption and observe that in general Random forest achieves higher average utilisation levels than UL. Random forest performs best in increasing the average utilisation in 5 out of 6 of the cumulative time windows, bringing up to 10.08% more utilisation for CPU and 10.39% for RAM when compared to UL. Furthermore, we notice that the actual peak predictor has, in the last two measurements, lower levels of utilisation than random forest or the default

Table 5.2: Waiting time and number of evictions for tasks of different priorities.

		Priorities		
		free	***normal***	***high***
Multiple LR	*WT≤2s*	96.43%	98.44%	95.23%
	Evictions	36467	78065	1633
	mean WT	0.0013s	**0.0001s**	**0.0010s**
Random Forest	*WT≤2s*	97.95%	98.98%	97.20%
	Evictions	17834	34209	169
	mean WT	**0.0013s**	0.0004s	0.0008s
User Limit	*WT≤2s*	**98.74%**	**99.46%**	**99.60%**
	Evictions	**11645**	**24606**	**25**
	mean WT	0.0016s	0.0007s	0.0020s

user limit predictor. The reason is that the two predictors and the UL have to deal with evictions, which produces an increase on the overall consumption of the cluster.

The figure also shows that in the case of CPU, there is a maximum difference of 20.17% for Multiple LR with regards to UL. In the case of RAM, the results are similar, namely a maximum difference of 21.36% for Multiple LR, when compared to UL. Moreover, we observed large standard deviations for the all the studied predictors. For instance, in the case of Random forest the average standard deviation for CPU is 21.68% and for RAM 22.29%.

In Table 5.2 we study the task waiting time and the number of evictions, while considering the priorities of the tasks. We highlighted with bold font the best performance for each of the cases considered. When studying the percentage of tasks with a waiting time (WT) lower or equal to 2 seconds, we noticed a difference between Random forest and UL of less than 0.8%, for tasks of low and normal priority. In the case of tasks with high priority, the difference is 2.26%. Moreover, when using UL, the scheduler produces the lowest number of evictions regardless of the priority of the task, this is expected since users tend to over-request the amount of resources that they need. Random forest and Multiple LR produce the lowest average waiting time for tasks that were not evicted, however the difference with regards UL is very small, since the maximum average WT is only 0.0013 seconds. Furthermore, given the high number of active machines and their lower levels of utilisation, using Multiple LR yields the highest number of evictions and lowest percentage of tasks being scheduled in less than 2 seconds.

The waiting time for evicted tasks is high for all the predictors. For instance, the average WT for evicted tasks with Normal priority is almost 10 minutes when the scheduler uses the Random forest predictor. The reason this large WT is that when

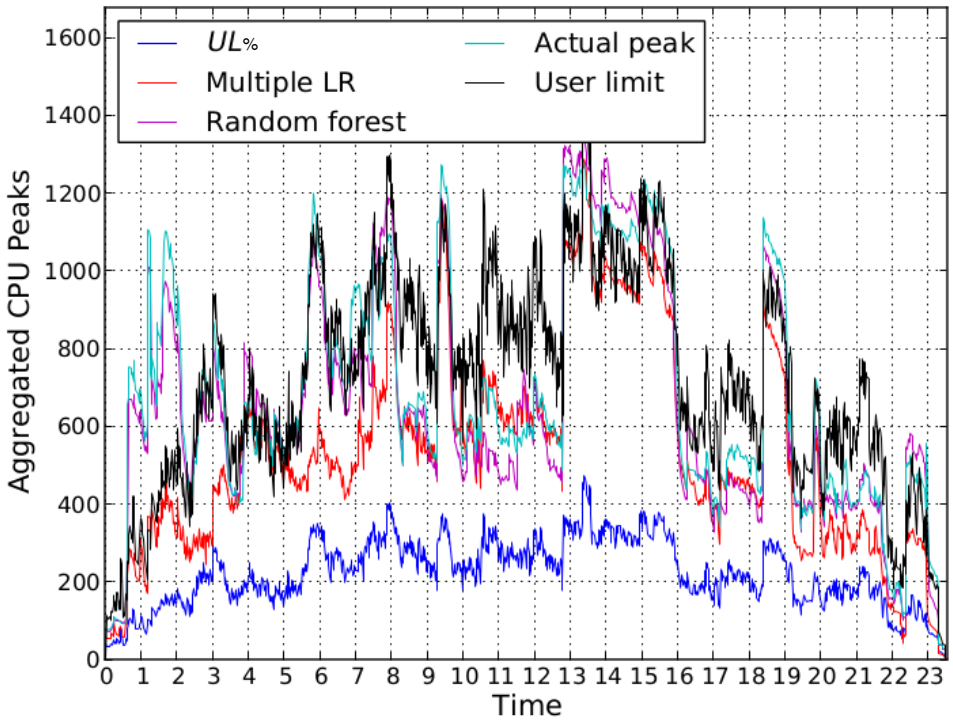


Figure 5.7: Aggregated CPU peaks.

our scheduler evicts a task, it only considers the task's priority, regardless of how long a task has been running.

To better understand the performance of the predictors, we aggregate the peak CPU utilisation for all the tasks with the assumption that the waiting time is 0 seconds. Figure 5.7 presents the results for all the prediction techniques in terms of aggregated CPU peaks. One can interpret the cumulated actual peaks as the amount of work that the cloud system has to tackle. For each predictor considered here, we report the cumulated peaks as expected by the predictor. Visually, the closer a predictor is to the actual workload, the better the predictor is. We notice that random forest is the predictor that resembles the most to the *Actual Peak*. Moreover, we notice that Multiple LR and $UL_{\%}$ reach very low levels of utilisation when compared to the *Actual Peak*. The random forest load predictor is the best candidate for accurate forecasts of peak resource utilisation.

5.5.3.3 Eviction policies

The last aspect investigated here is the behaviour of the evictions heuristics discussed in Section 5.4. As we mentioned in the previous section, when introducing uncertainties in the problem, one cannot guarantee that there will be no evictions. This is simply due to the fact that tasks may not behave as the predictor's expectations leading to a possibly overloaded machine. Figure 5.8 shows the number of evictions while using the random forest predictor and the restricted random scheduling policy. The figure distinguishes the three classes of priority (i.e. low, normal and high), the lower the number of evictions, the better the eviction policy is.

As can be seen in the figure, evicting tasks based on their priority level seems to be a sensible policy to minimise the overall number of evictions. It can be assumed that tasks with higher priorities are also the ones that have the longest duration thus the behaviour of both *min priority* and *min running time* policies are rather similar.

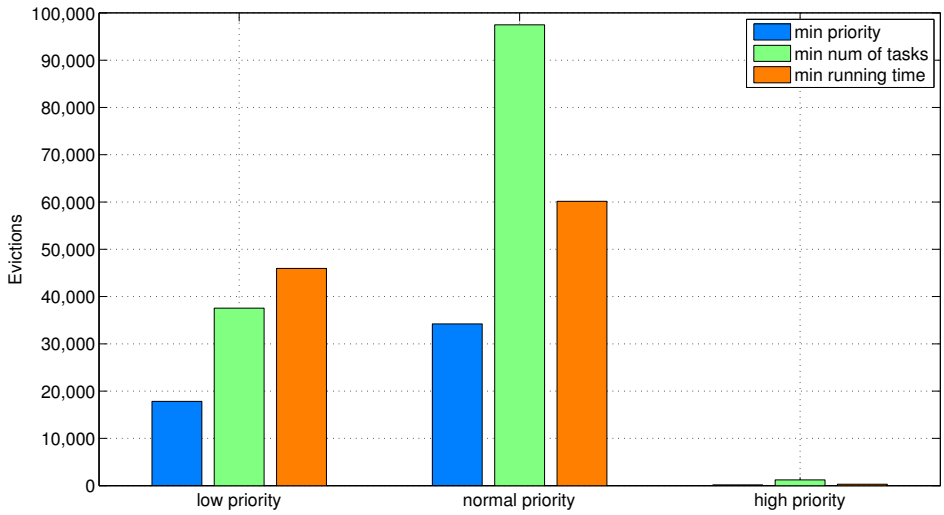


Figure 5.8: Performance of the eviction policies across different classes of priority. Tasks were scheduled with the restricted random policy couple with the random forest predictor.

Furthermore, we can see that trying to locally minimise the number of evictions with the *min num tasks* yields the highest number of eviction. This suggests that information available locally on a machine is too partial to minimise the number of evictions globally.

5.6 Conclusion and Limitations

In this chapter we proposed a method that addresses the problem of online scheduling of tasks under several operational constraints. The problem is to efficiently allocate tasks with multiple resource requirements on heterogeneous machines. The objective is to maximise the utilisation of the machines while minimising the waiting time of the tasks. Although, the estimated peak resource requirements of these tasks are provided by users, the actual peak requirements can vary significantly. Mostly, these requirements are over-estimated resulting in allocating more resources which results in the poor performance of the cluster. Also, when these requirements are under-estimated, cumulative requirements of the tasks running on a machine can exceed its capacity. Consequently one or more tasks need to be evicted, which are then again scheduled on the cluster leading to the poor performance of the cluster.

We further studied the impact of different prediction techniques when scheduling and evicting tasks using different heuristics. We discovered that reductions in resource prediction errors do not always bring benefits. This is because of the large number of task evictions caused by continuously under provisioning the amount of resources task would need in order to reduce the estimation errors. Moreover, this causes an increase in the task waiting time and a decrease in the overall cluster utilisation, since new machines need to be activated in order to handle the evicted tasks. Our results show that the Random forest predictor significantly surpasses the default user limit for almost all metrics considered, bringing a reduction in terms of number of active machines by up to 3,337, and increasing the average cumulative utilisation of the cluster by 7.72% for both resources considered, CPU and RAM.

Overall, the best configuration of our methodology for the studied trace was observed when predicting resource requirements using random forest, scheduling using restricted version of round robin and evicting tasks based on their priorities. As claimed in sub-thesis 2, discussed in Section 1.4, machine learning can be used to, at least partially, produce more accurate input data for the subsequent optimisation model. This leads to an overall better consolidation behaviour in this complex real-world cloud setting.

In this chapter, we modeled the problem of online workload consolidation as a constrained scheduling problem. Although we investigated the dynamics of evicting and rescheduling tasks, we did not capture the possibility, often desirable, to allow tasks' executions to be paused, the task migrated to another machine and finally resumed. This is the object of the next chapter for which task migrations are allowed.

Chapter 6

Proactive Consolidation with VM Migrations

Summary. *In this chapter we explore a relaxed workload consolidation problem in which tasks are allowed to be migrated from a physical host to another one. Known as dynamic workload consolidation, the optimisation challenge tackled here is to keep servers well utilised by deciding which virtual machines to migrate, where to migrate, when to migrate, and, when and which servers are to be switched on/off. Achieving this goal optimally requires the capability of predicting the future time-variable resource demands of VMs accurately and computing the plan for migrating VMs for efficient workload consolidation quickly. We developed an optimisation model coined the Proactive Workload Consolidation Problem (PWCP). Solving the problem as a monolithic offline problem with infinite time windows is impossible due to the impractibility of forecasting demands and the intractability of finding optimal assignments of VMs to servers. We formulate the PWCP in a more realistic way by defining a time window of a given size in which the information is known accurately and solve a, possibly infinite, sequence of optimisation problems moving forward in time.*

In this chapter, we discuss a consolidation problem that can be classified as a dynamic consolidation problem. This is due to the nature of the workload under consideration. Tasks forming the workload considered in this chapter are subject to time varying resource requirements. More importantly, tasks may be migrated from a host machine to another one in order to improve consolidation. The wide adoption of cloud solutions

has raised the challenge maintaining control over the Quality-of-Service (QoS) experienced by end-users. The computing resources provided by cloud operators must be accessible at all time and must guarantee a certain level of performance. To that end, both parties contractually agree on terms defining the service level agreements (SLAs). More information on QoS can be found in [SBK+16].

The online and dynamic nature of the problem raises challenges linked to partial information at decision time. One of the questions we investigate is how far one is required to look ahead in terms of the number time-periods and still retain the minimum energy cost of a given horizon without violating Service Level Agreements (SLAs).

We perform investigations to understand the relationship between the number of time-periods considered in one optimisation step and migration limits on the SLAs, energy cost, server transition cost and migration cost. The advantages of answering this question are twofold: (1) The size of the optimisation problem reduces as the number of time-periods reduces, which can help in finding good quality solutions quickly; (2) The overall accuracy of the forecasting technique increases as the length of the horizon reduces, which can help in finding more reliable solutions. Therefore, we aim at studying the trade-off between the quantity of locally available information versus the quality of a dynamic workload consolidation policy.

We perform an empirical study to understand the relationship between input parameters such as the number of time-periods in a single optimisation step and migration-limits (impacting the QoS) on the outputs of workload consolidation over a given time-horizon like the number of SLAs violations, energy cost, server-states transition cost, and migration cost. Our results suggest that if we look ahead by only a few time-periods then it can lead to significantly more efficient resource utilisation over the entire horizon and consequently higher energy efficiency and very few service level violations.

6.1 The Proactive Workload Consolidation Problem

The Proactive Workload Consolidation Problem (PWCP) aims to assign a server to each VM based on not only their current demands but also their future demands over upcoming time-periods. The objective is to minimise the energy cost over a given time horizon without violating SLAs. Ideally one would like to make sure that this holds for all time-periods up to h, where h represents an arbitrary large number of time-periods. Therefore, the aim is to find the right value of w that allows us to achieve

that by solving a sequence of optimisation problems moving forward in time. In the following, we introduce an optimisation model for the PWCP.

Let $V = \{v_1, \ldots, v_n\}$ be the set of VMs, $S = \{s_1, \ldots, s_m\}$ be the set of servers and $T = \{p_1, \ldots, p_h\}$ be the set of time-periods. Here p_0 represents the time-period preceding decision time, p_1 is the current time-period for which we want to assign VMs to servers, and p_2 to p_w, where $w \leq h$, are the future time-periods which we want to consider now to plan for future migrations. We must guarantee that the VMs have enough resources at each time period.

The actual prediction of the resource demands of VMs is out of the scope of this chapter. We, therefore, use the historical resource requirements of the VMs that originate from the Green Data Centre of Business & Decision Eolas located in Grenoble, France, which deals with web applications, e-commerce, e-business, e-administration, etc, where the CPU usage of a VM changes over time while the memory usage is constant over time. Detailed information on the data set can be found in [CMOS13a].

Virtual Machines. A VM v_i is characterised by memory consumption M_{it} and CPU consumption U_{it} at time-period t, a set $A_i \subseteq S$ of allowed servers where it can be hosted, and a potential initial server denoted by $Iserv_i$. While solving the problem at time t, v_i has been allocated the $Iserv_i$ by the decision taken in the previous optimisation step.

Servers. A server s_j can be in two different states: ON or STBY (stand-by). It is characterised by:

- A CPU capacity $Umax_j$ and a memory capacity $Mmax_j$.
- A fixed cost of usage $Emin_j$ (in Watt) when the server is ON.
- A unit cost τ_j per unit of CPU usage.
- A basic CPU consumption Ca_j when it is ON to run the operating system and other permanent tasks.
- An energy consumption $Esby_j$ when it is in state STBY.
- An energy consumption $Esta_j$ to change the state of servers from STBY to ON.
- An energy consumption $Esto_j$ to change the state of servers from ON to STBY.
- A maximum number $Nmax_j$ of virtual machines that can be allocated to it at any time-period.
- A set of periods $P_j \subseteq T$ during which s_j is forced to be ON.

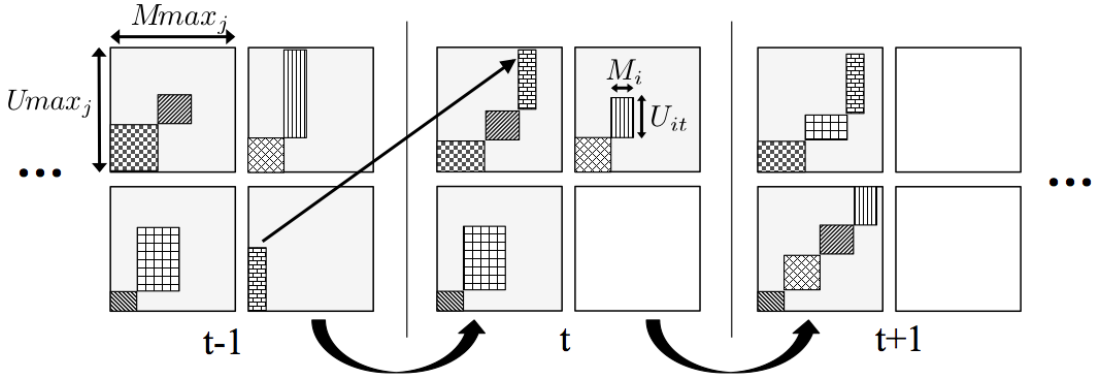


Figure 6.1: A solution of PWCP over three time-periods

- A potential initial state $Istate_j \in \{0, 1\}$.

If a server is ON, its minimum cost is $Emin_j + \tau_j Ca_j$. Therefore, for the sake of simplicity, to compute the fixed energy cost of an active server we include the basic consumption Ca_j in $Emin_j$ and denote that by $Emin'_j = Emin_j + \tau_j Ca_j$. We also shift the CPU capacity of a server and denote that by $Umax'_j = Umax_j - Ca_j$.

Migrations. The maximum number of changes of servers amongst all virtual machines from one time-period to the next is denoted by N and the cost of a migration by $Cmig$.

This problem can be seen as a series of packing problems, one per time period, in two dimensions (CPU and memory). These packing problems are coupled by both the migration constraints and the cost for changing the state of a server. Figure 6.1 gives an overview of the problem. This example has four servers, each shown by a rectangle whose dimensions represent the CPU and memory capacities of that server. A VM is a small rectangle whose height (its CPU) varies from one period to the next. Therefore, the sum of the heights (CPU) must fit within the capacity of the server they are assigned to. In this scenario, the CPU needs of some virtual machines decreases allowing us to find better packings and possibly turn off two servers at $t + 1$.

6.2 An Integer Linear Model for the PWCP

We reformulate the integer linear model of the PWCP that was first introduced in [CMOS13b] in which the following variables are used: $x_{ijt} \in \{0, 1\}$ indicates whether virtual machine v_i is placed on server s_j at time t. $cpu_{jt} \in [0, Umax'_j]$ gives the CPU consumption of s_j at period t. $o_{jt} \in \{0, 1\}$ is set to 1 if s_j is ON at time t, 0 otherwise. $bto_{jt} \in \{0, 1\}$ is set to 1 if s_j was in STBY at $t - 1$ and is turned ON at t. $otb_{jt} \in \{0, 1\}$ is set to 1 if s_j was in ON at $t - 1$ and is put STBY at t. $a_{it} \in \{0, 1\}$ is

set to 1 if v_i is on a different server at t than the one it was using at $t-1$. The model is summarised in Model (6.1).

Minimize $\sum_{s_j \in SE} \sum_{t \in T} (Esta_j bto_{jt} + Esto_j otb_{jt} + \tau_j cpu_{jt} + Emin'_j o_{jt}) +$
$Cmig(\sum_{v_i \in VM} \sum_{t \in T} a_{it})$

$$
\begin{array}{lll}
(6.1.1)\ \sum_{s_j \in SE} x_{ijt} & = 1 & (\forall\, v_i \in VM, p_t \in T) \\
(6.1.2)\ x_{ijt} & = 0 & (\forall\, v_i \in VM, p_t \in T, s_j \notin SA_i) \\
(6.1.3)\ x_{ijt} & \leq o_{jt} & (\forall\, v_i \in VM, p_t \in T, s_j \in SE) \\
(6.1.4)\ cpu_{jt} & = \sum_{v_i \in VM} U_{it} x_{ijt} & (\forall\, s_j \in SE, p_t \in T) \\
(6.1.5)\ cpu_{jt} & \leq Umax'_j o_{jt} & (\forall\, s_j \in SE, p_t \in T) \\
(6.1.6)\ \sum_{v_i \in VM} M_{it} x_{ijt} & \leq Mmax_j o_{jt} & (\forall\, s_j \in SE, p_t \in T) \\
(6.1.7)\ \sum_{v_i \in VM} x_{ijt} & \leq Nmax_j o_{jt} & (\forall\, s_j \in SE, p_t \in T) \\
(6.1.8)\ a_{it} & \geq x_{ijt} - x_{ijt-1} & (\forall\, v_i \in VM, s_j \in SE, p_t \in T) \\
(6.1.9)\ \sum_{v_i \in VM} a_{it} & \leq N & (\forall\, p_t \in T) \\
(6.1.10) bto_{jt} & \geq o_{jt} - o_{jt-1} & (\forall\, s_j \in SE, p_t \in T) \\
(6.1.11) otb_{jt} & \geq o_{jt-1} - o_{jt} & (\forall\, s_j \in SE, p_t \in T) \\
(6.1.12) o_{jt} & = 1 & (\forall\, s_j \in SE, p_t \in P_j) \\
(6.1.13) x_{ij0} & = 0 & (\forall\, v_i \in VM, s_j \in SE - \{Iserv_i\}) \\
(6.1.14) x_{i,Iserv_i,0} & = 1 & (\forall\, v_i \in VM) \\
(6.1.15) o_{j0} & = Istate_j & (\forall\, s_j \in SE)
\end{array}
\tag{6.1}
$$

Constraint (6.1.1) states that a VM has to be on a server at any time; Constraints (6.1.2) enforces the forbidden servers for each VM; Constraints (6.1.3) enforces a server to be ON if it is hosting at least one VM; Constraints (6.1.4) links the CPU load of a server to the VMs assigned to it. Constraints (6.1.5–6.1.7) are the resource constraints (CPU, memory and cardinality) of each server; Constraints (6.1.8,6.1.9) allow us to count the number of migrations and state the limit on N; Constraints (6.1.10,6.1.11) keeps track of the change of states of the servers; Constraints (6.1.12) states the periods where a server has to be ON; Finally constraints(6.1.13–6.1.15) enforce the initial state ($t = 0$). The number of constraints of this model is dominated by the $n \times m \times h$ number of constraints $(6.1.8)$ and constraints $(6.1.3)$.

To cope with the dynamical nature of the problem, sub-models extracted from Model (6.1) Mod will be repeatedly solved within a fixed horizon window with size $w < T$. We use $\texttt{Mod}_1^h$ to denote the problem's formulation (6.1). We use $\texttt{Mod}_t^{t+w}$ to denote a restriction of the original model on a set of consecutive time periods between $\{t \,..\, t+w\}$. For each time-period $t \in \{0 \,..\, h-w\}$, $\texttt{Mod}_t^{t+w}$ is solved within a fixed horizon window with size $w < T$. At each iteration the current time period is shifted by one which eventually leads to a feasible solution for the original model.

This myopic way of solving the original model has drawbacks:

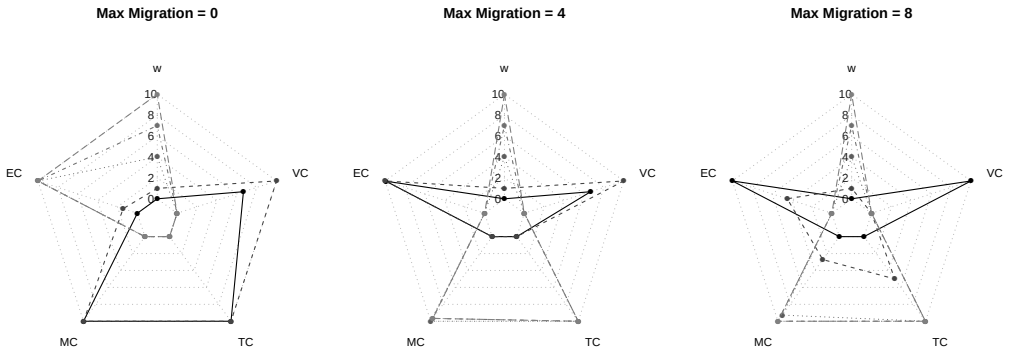


Figure 6.2: Varying window size w over selected values of N. Each value of $w \in \{0, 2, 4, 6, 8, 10\}$ corresponds to a different line style.

1. There is no opportunity to prove that the computed solution of the original model is optimal. We rather approximate optimality on an infinite time line.
2. It can lead to non-feasible restrictions of the original problem. This is mainly due to the fact that the feasibility of $\texttt{Mod}_t^{t+w}$ largely depends on the solution of $\texttt{Mod}_{t-1}^{t-1+w}$. In order to deal with infeasible restrictions of the problem, one must relax the SLA constraints, in order to be able to compute a valid assignment from VMs to servers.

Relaxing the SLA constraints can be done in two ways. The first approach is to set the maximum number of migrations to $|V|$ allowing one to migrate as many VMs as needed. The other approach is to relax the maximum workload that a server can handle. It is clear that these two approaches impact the quality of service delivered to clients respectively by saturating the network within the data centre, or overloading server capacity over one or several time periods. For the sake of conciseness only the former strategy will be presented here.

6.3 Empirical Analysis

We investigate the impact of the tightness of the migration constraint and the length of the selected time-window on the outcome of the PWCP. We vary the values of w, i.e. the number of time periods considered in a single optimisation step, and the value of N that is impacting the level of tightness of the SLAs constraint. Note that every different value of N gives rise to an instance of the problem with possibly a very different optimal solution.

Finally in order to evaluate the performance of the optimisation process for each of the combination of (w, N) we compute electricity Cost (EC), migration cost (MC) and transition cost (TC) - induced over the complete full time horizon. In addition, since the iterative solving process can lead to infeasibility, we also compute the proportion of times SLAs were violated (VC). These different metrics can be read on a star plot for which each line can be thought of as a measure of the various features of a solution given by the couple (w, N).

We first illustrate the impact of sequentially solving PWCP with different window sizes. Figure 6.2 shows the performance in terms of cost and SLA violations of solutions computed under various window size regimes ($w \in \{0, 2, 4, 6, 8, 10\}$). Each of these regimes is represented with a different line style on the figure. It can be observed that when the VMs are assigned to servers without any knowledge about future requirements (i.e., $w = 0$), it performs badly in terms of EC in most of the cases. It

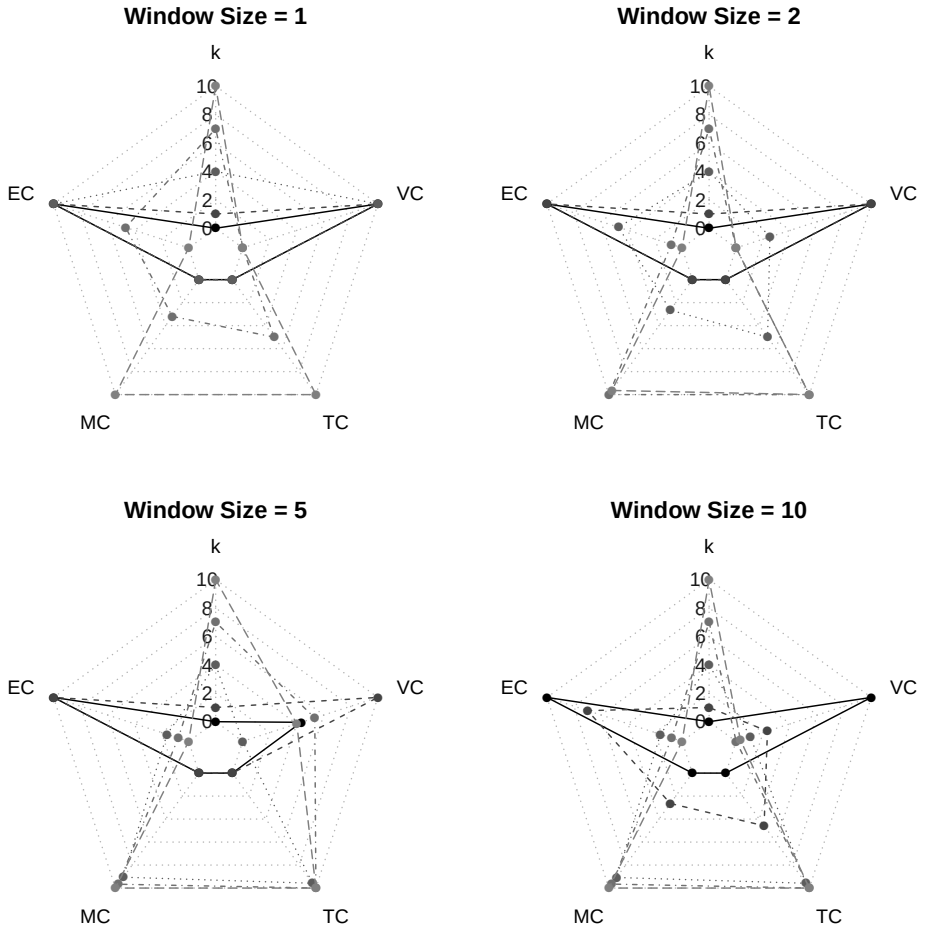
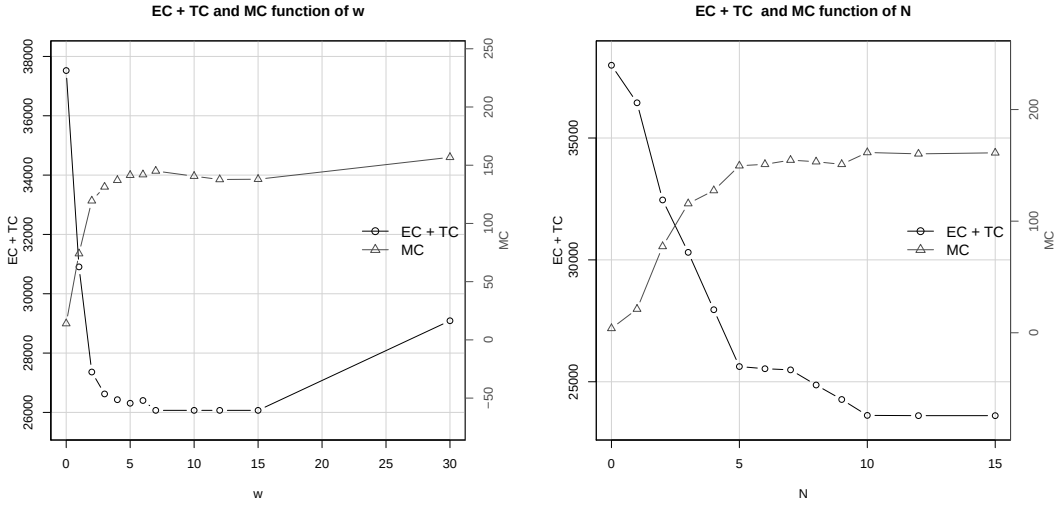


Figure 6.3: Varying the migration limits (k) over selected values of w. Each value of $k \in \{0, 2, 4, 6, 8, 10\}$ corresponds to a different line style.

can also be observed that, in general, introducing more time periods in one optimisation step helps in finding solutions with a lower EC. This is due to the fact that the more time periods one considers in one optimisation step, the more *future-aware* the solution will be.

Figure 6.2 shows that both parameters N and w have an impact on the proportion of optimisation steps leading to non-satisfiable models (VC). On the one hand, it is harder to find a solution to a very tight problem (i.e. small values of N), and on the other hand, restricting the window size (i.e. small values of w) makes the optimisation process very myopic to future changes and might drive the solution towards infeasible regions. One should also note that, for $N = 0$ and $w = 0$, the problem is too constrained to find solutions without breaking the migration constraint. As previously noted, when this is the case, we relax the constraint for the current time period allowing VMs to move dramatically to other servers. We thus observe that this case performs very well in terms of electricity cost and shows higher migration and transition costs.

To get a better understanding of the trade-off between energy cost of a solution and quality of service, in Figure 6.4a and Figure 6.4b one can see values of the aggregation of energy and transition cost (EC + TC) and the migration cost function of the number of time periods and the maximum number of migration respectively. The figures reported here are an aggregation of local optima as suggested in Section 6.2. If one thinks of the migration cost as a good indicator of the QoS levels, then we see clearly that these two component are somehow conflicting with each other. As the number



(a) Split of the costs function of w

(b) Split of the costs function of N

Figure 6.4: Trade-off between energy cost of a solution and Quality of Service.

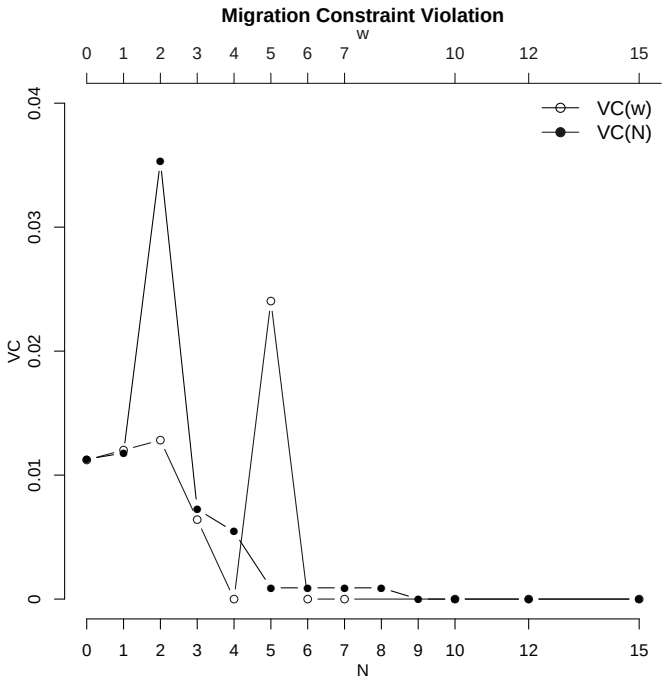


Figure 6.5: VC function of N and w

of time periods w (resp. N) rises, we observe that the solutions are showing dramatic improvements in terms of electricity cost. This is due to the opportunity to build solutions in which VMs are more easily moved from one server to another one. On the downside, if the solution admits too many migrations users may experience degraded quality of service.

The last aspect discussed in this section is the number of times the solving process has to relax the migration constraint over the various setting of the couple (w, N). Figure 6.5 suggests that iterative solving violates the migration constraints up to 4% of the time restricted models. These figures are respectively reached for small values of both parameters. For problems less tightly constrained or more *future aware*, the risk to violate the migration constraint tends to zero, guaranteeing a proper QoS level.

6.4 Conclusion and Limitations

In this chapter we have presented a formulation of the Proactive Workload Consolidation Problem. This problem differs from traditionally studied workload consolidation problems due to its dynamic nature and its incomplete information at decision time. We developed a model accounting for these aspects and investigated various combination of both the size of the window in which the optimisation process takes place and

the tightness of the migration constraint. We have found that by looking only a few time periods ahead, it is possible to achieve significant reductions in terms of energy cost if the problem is not too tightly constrained. As a downside, we have shown that if too few time periods are considered within the window, the optimisation process locally drives the solution towards infeasible solutions. This is in particular true for tight problems for which the user might experience lower QoS.

There are also other aspects that one needs to consider when dealing with predicted demands for multiple time-periods. If the predicted demands of virtual machines are over-estimated then in the planning phase it might not be possible to compute feasible solutions even though there might exist one when the actual demands are known.

Chapter 7

On Bin Packing Instances

Summary. *This chapter shifts the focus on generating bin packing instances that can be used to further study and generalise previous results. We present a benchmark generator for bin packing instances based on the well-known Weibull distribution. Using the shape and scale parameters of this distribution we can generate benchmarks that contain a variety of item size distributions. We show that real-world bin packing benchmarks can be modeled extremely well using our approach. We also study both systematic and heuristic bin packing methods under a variety of Weibull settings. We observe that for all bin capacities, the number of bins required in an optimal solution increases as the Weibull shape parameter increases. However, for each bin capacity, there is a range of Weibull shape settings, corresponding to different item size distributions, for which bin packing is hard for a CP-based method.*

The motivation for the work presented in this chapter comes from the variety of packing related problems studied in previous chapters. Indeed, the various workload consolidation problems discussed in this dissertation far have a rather strong connection to the bin packing problem. In previous chapters, we have used data coming from real-world applications of cloud systems to benchmark approaches to solve these problems. In this chapter, we wish to open a discussion on generating synthetic benchmarks for the one dimensional bin packing problem. In turn, these could be used to further analyse the various problems presented in this dissertation.

While there are many benchmark suites for bin packing in the literature [Fal96, Kor03, SKJ97, SW97, SW98, WG96], these are all artificial and lacking a practical basis. Typically, as for example in the benchmarks by Scholl and Klein [SKJ97], item sizes

are generated using either uniform or normal distributions. As Gent has pointed out, current benchmark suites in this area are often unrealistic and trivial to solve [Gen98]. This claim also is found in a paper by Richard E. Korf [Kor02]. Regin et al. [RR11] have called for more realistic suites for use in studying large-scale data centre problems. It is this requirement that this chapter seeks to address.

Section 7.1 presents a parameterisable benchmark generator for bin packing instances based on the Weibull distribution [Wei51]. Using the shape and scale parameters of this distribution a variety of item size distributions can be generated. In Section 7.2 we show that a number of real-world bin packing benchmarks can be modeled well using this approach. We study the behaviour of both systematic (Section 7.3) and heuristic (Section 7.4) bin packing methods under a variety of settings. We show that our framework allows for very controlled experiments in a bin packing setting in which the distribution of item sizes can be precisely controlled. We discuss how the difficulty of bin packing is affected by the item size distribution and by bin capacity. Specifically, we observed that for all bin capacities, the number of bins required in an optimal solution increases as the Weibull shape parameter increases. However, for each bin capacity, it seems that there is a range of Weibull shape settings, corresponding to different item size distributions, for which bin packing is hard for a CP-based method.

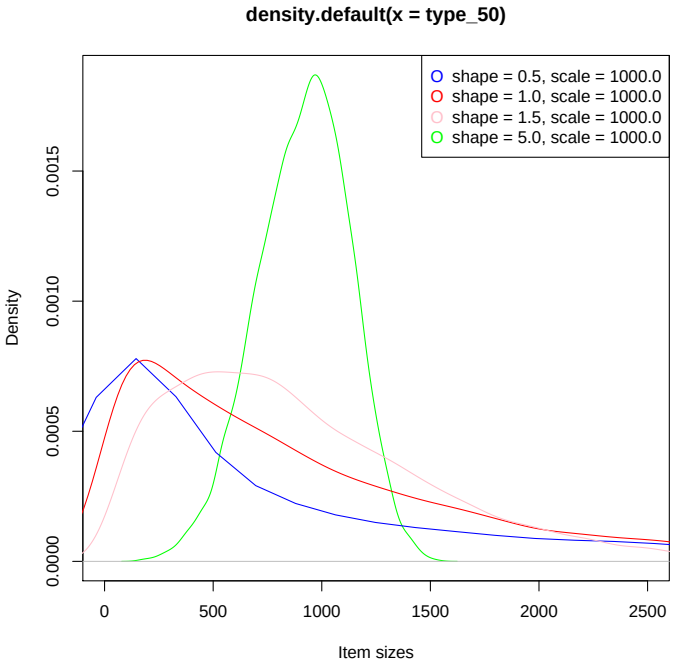
7.1 The Weibull Distribution

In probability theory, the Weibull distribution is a continuous probability distribution. It is named after Waloddi Weibull, who presented the distribution in a seminal contribution in 1951 [Wei51]. The Weibull distribution is defined by a shape parameter, $k > 0$, and a scale parameter, $\lambda > 0$. The probability density function, $f(x; \lambda, k)$, of a random variable x distributed according to a Weibull distribution is defined as follows:

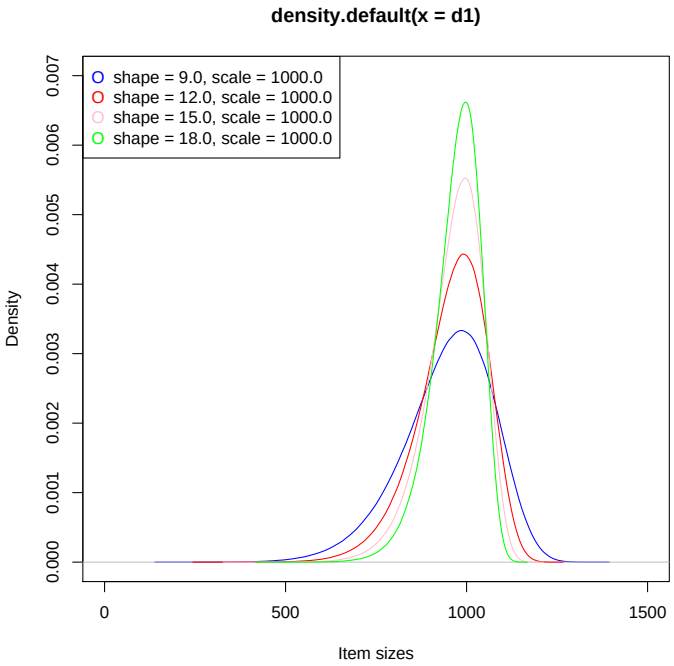
$$f(x; \lambda, k) = \begin{cases} \frac{k}{\lambda} \cdot (\frac{x}{\lambda})^{k-1} \cdot e^{-(x/\lambda)^k} & x \geq 0, \\ 0, & \text{otherwise.} \end{cases}$$

The Weibull distribution can model many situations that naturally occur in a variety of problem domains involving distributions of time horizons, time slots or lot sizes [Wei51]. Figures 7.1a and 7.1b present several examples of different distributions that can be obtained by instantiating the Weibull distribution. Figure 7.1a presents four different distributions for small values of the shape parameter, k. Clearly very different regimes are possible, some exhibiting extremely high skew around the value specified

by the distribution's scale parameter, λ. In Figure 7.1b, larger values of the shape parameter are considered.



(a) Small shape parameters



(b) Larger shape parameters

Figure 7.1: Weibull distributions

In this chapter we propose using the Weibull distribution as the basis for a parameterisable benchmark generator for bin packing instances in which the *item sizes* are generated according to a Weibull distribution parameterised by specific values of k and λ. Using these parameters a variety of item size distributions can be generated. In Section 7.2 we show that some real-world bin packing benchmarks can be modelled extremely well using a Weibull distribution.

7.2 Fitting Weibull Distributions to Real-world Instances

In this section we demonstrate the flexibility of the Weibull distribution in fitting to a variety of bin packing problems coming from real-world applications. In Section 7.2.1 we show a specific example of how well the Weibull distribution can fit to a problem instance arising from the 2012 ROADEF/EURO Challenge. This example will show, visually, the quality of the fit that can be obtained. However, in Section 7.2.2 we present a more rigorous analysis of the goodness-of-fit that can be achieved through the use of two standard statistical tests.

7.2.1 An Example Problem in Data Centre Management

The 2012 ROADEF/EURO Challenge[1] is concerned with the problem of machine reassignment, with data and sponsorship coming from Google. The subject of the challenge is to find a best-cost mapping of processes, which have specific resource requirements, onto machines, such that a variety of constraints are satisfied. A core element of the problem are bin packing constraints stating that the total amount of a given resource required by the processes assigned to a machine does not exceed the amount available.

An important element of this challenge is the mapping of processes to machines such that the availability of each resource on the machine is not exceeded by the requirements of the set of services assigned to it. This subproblem is a multi-capacity bin packing problem: each machine is a bin with many elements defined by the set of resources available, and each process corresponds to an item that consumes different amounts of each resource.

[1] `http://challenge.roadef.org/2012/en/index.php`

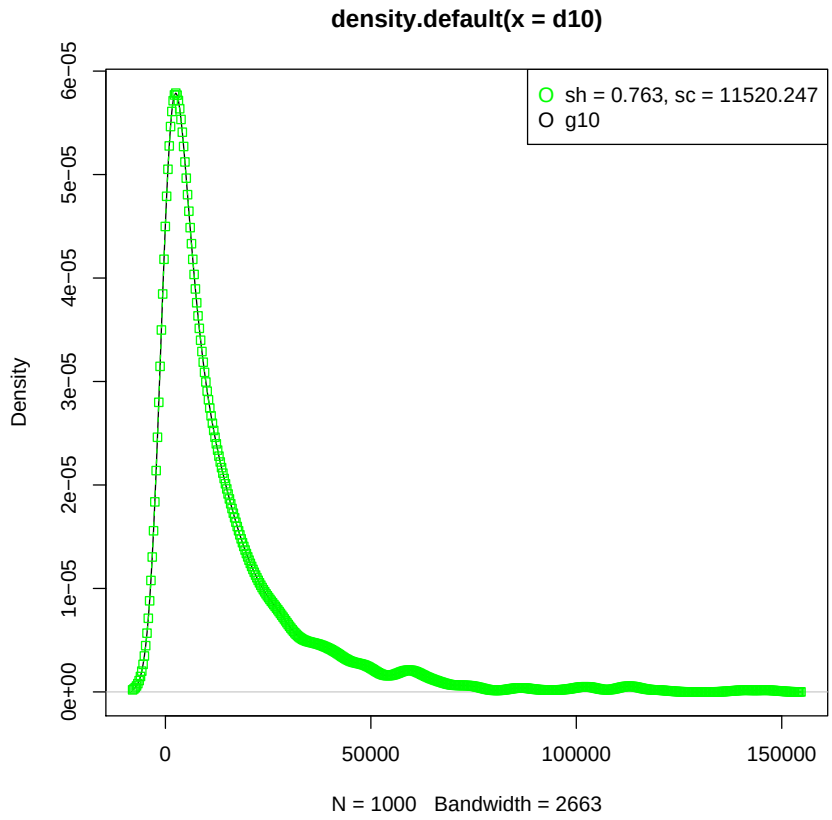


Figure 7.2: An example of the quality of fit one can achieve when using a Weibull distribution for a real-world bin packing problem. Here we present the data and Weibull fit associated with Resource 10 of instance `a2(5)` from the 2012 ROADEF/EURO Challenge sponsored by Google.

Figure 7.2 presents an example probability distribution for Resource 10 from instance `a2(5)` of the benchmarks available for the ROADEF/EURO Challenge. The probability density function that corresponds to the actual data is plotted as a line. We can see clearly that this distribution is extremely skewed, with the majority of the probability mass coming from smaller items. Another characteristic of the data is the spread along the x-axis, showing that the range of likely item sizes spans several orders-of-magnitude, and there is a very small possibility of encountering extremely large items.

We used R, the open-source statistical computing platform [The], to fit a Weibull distribution to this data, using maximum likelihood fitting [Mar]. Specifically, we have used the R Weibull Distribution Maximum Likelihood Fitting implementation by Wessa, which is available as an online service [Wes]. The resulting Weibull is presented in Figure 7.2 as the circles imposed on the density function from the data. By observation we can see that the fit is extremely good. In the next section we will study the quality of fit more rigorously, demonstrating that it is statistically significant.

7.2.2 Verifying the Goodness-of-Fit

We study a variety of benchmark bin packing problems. As mentioned above, the 2012 ROADEF/EURO Challenge provides a publicly available set of problem instances that contains many bin packing instances. In addition to those, we consider real-world examination timetabling benchmarks. The bin packing component of these problems involves scheduling examinations (items) involving specified numbers of students (item sizes), into rooms of specified capacity (bin capacities) within time-slots (number of bins). We consider the data sets available from universities in Toronto, Melbourne and Nottingham [QBM+09]. These are available from the OR library.[2]

We used two goodness-of-fit tests to evaluate whether or not the Weibull distribution is capable of modeling the distribution of item sizes in these data sets. We discuss each of these tests in the following sections.

The Kolmogorov-Smirnov Test. The two-sided Kolmogorov-Smirnov (KS) test is a non-parametric test for the equality of continuous, one-dimensional, probability distributions[3]. As implemented in R, this test requires two sample sets: one representing the observed data, and the other representing a sample from the hypothesised distribution. In our setting, the observed data is represented by the item sizes from the benchmark we wish to model, while the second set is a vector of items generated according to the best-fit Weibull distribution with parameters (shape and scale) estimated from the observed data [Ric05]. The null hypothesis of this statistical test is that *the two data sets come from the same underlying distribution*. For a 95% level of confidence, if the p-value from the test is at least 0.05, then we cannot reject the null hypothesis.

The KS test was performed on all instances from our exam timetabling (ETT) and ROADEF/EURO benchmark suites; the details of a randomly selected subset are presented in Table 7.1. We can see that most of the ETT item size distributions can be accurately modeled by a Weibull distribution since the corresponding p-values are above 5% (highlighted in bold). However, the KS test clearly rejects the null hypothesis for the ROADEF/EURO instances, most likely due to a both the size of the data sets and the presence of outliers in the tail of the distribution.

It is known that when dealing with large data sets with a small number of large outliers, this test tends to underestimate the p-value. This means that even if the null hypothesis is rejected the candidate distribution might still characterise the data set [MLDMD11].

[2] `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`
[3] `http://mathworld.wolfram.com/Kolmogorov-SmirnovTest.html`

For this reason we use the χ^2 test to further validate the results.

The χ^2 Test. As a complementary approach, we used the χ^2 goodness-of-fit test which is less sensitive to outliers in the sample data.[4] The null hypothesis is that the observed and expected distributions are not statistically different.

The procedure requires grouping items into γ categories according to their size. Based on these categories, we can compute the expected number of values in each category, assuming that the item sizes are drawn from a Weibull distribution with shape and scale parameters estimated from the data set. The χ^2 statistic is then computed as:

$$\chi^2 = \sum_{i=1}^{\gamma} (O_i - E_i)^2 / E_i,$$

from which we can obtain the corresponding p-value, where O_i and E_i are the observed and expected frequencies of each category i, respectively. We model the tail of the distribution, in the standard way, by building a wider category that counts all items in the tail of the distribution. The other $\gamma - 1$ categories are equally sized.

As shown in Table 7.1, the null hypothesis cannot be rejected for any of the benchmarks that are presented. Therefore, the conclusion is that the Weibull distribution provides a good fit for the item size distributions in the benchmark instances we considered. We conjecture that it will also do so in very many other cases encountered in practice.

7.3 Systematic Search for Bin Packing

We consider the performance of a systematic constraint-based bin packing method on a wide number of classes of Weibull-based bin packing benchmarks. Our experiment involved varying the parameters of the Weibull distribution so that item sets for bin packing instances could be generated. A range of bin capacities were studied. The details of the experimental setup are described in Section 7.3.1.

7.3.1 Bin Packing Instances and Solver

We considered problems instances involving 100 items. We fixed the scale parameter, λ, of the Weibull to 1000. As experimental parameters we varied both the capacity of

[4] `http://mathworld.wolfram.com/Chi-SquaredTest.html`

Table 7.1: The parameters of the best-fit Weibull distributions obtained for randomly selected instances of a number of real-world examination timetabling benchmarks.

		Weibull Best-fit		KS test	χ^2 test		
Set	Instance	shape	scale	p-value	#(cat)	lbTail	p-value
ETT	Nott	1.044	43.270	**0.7864**	7	100	**0.059**
	MelA	0.946	109.214	**0.091**	10	427	**0.073**
	MelB	0.951	117.158	**0.079**	5	47	**0.051**
	Cars	1.052	85.438	0.037	18	53	**0.109**
	hec	1.139	138.362	**0.436**	10	293	**0.204**
	yor	1.421	37.049	**0.062**	7	117	**0.068**
RAODEF	$a1_3^2$	0.447	104,346.70	0.005	30	163,000	**0.105**
	$a1_3^3$	0.549	88,267.85	0.001	15	54,800	**0.068**
	$a2_1^5$	0.562	67,029.83	0.000	30	470,000	**0.768**
	$a2_4^4$	0.334	103,228.30	0.001	30	500,000	**0.051**
	b_6^3	0.725	40,469.74	0.000	20	185,000	**0.060**
	b_3^5	0.454	91,563.28	0.000	30	140,000	**0.088**

the bins and the shape of the Weibull distribution, generating 100 instances for each combination of parameters. The capacities we considered were $c \times max(I)$, where $c \in [1.0, 1.1, \ldots, 1.9, 2.0]$ and $max(I)$ is the maximum item size encountered in the instance. Therefore, the capacity of the bins considered were at least equal to the largest item, or at most twice that size.

For the shape parameter of the Weibull we considered a very large range: $[0.1, 0.2, \ldots, 19.9]$, yielding 199 settings of this parameter. By fixing the scale parameter to 1000 we considered item sizes that could span over three orders-of-magnitude. To build our problem generator we used the Boost library [Boo]. This is a `C++` API that includes type definitions for random number generators and a Weibull distribution, which is parameterized by the random number generator, the shape and the scale. Iteration capabilities for traversing the distribution of generated values are also provided. We generated 100 instances for each combination of shape and scale, giving 199 classes of item sets, providing 19,900 item sets. For each of these sets we generated bin packing instances by taking each set and associating it with a bin capacity in the range described above. In this way we could be sure that as we changed bin capacity, the specific sets of items to be considered was controlled.

Constraint-based bin packing Model. For our experiments we have used Gecode 3.7.0 [Gec06]. The bin packing model used is the most efficient one included in the Gecode distribution for finding the minimum number of bins for a given bin packing instance [STL12]. This model employs the L_1 lower bound on the minimum number of bins by Martello and Toth [MT90a].

It uses an upper bound based on the first-fit bin packing heuristic which packs each item into the first bin with sufficient capacity.

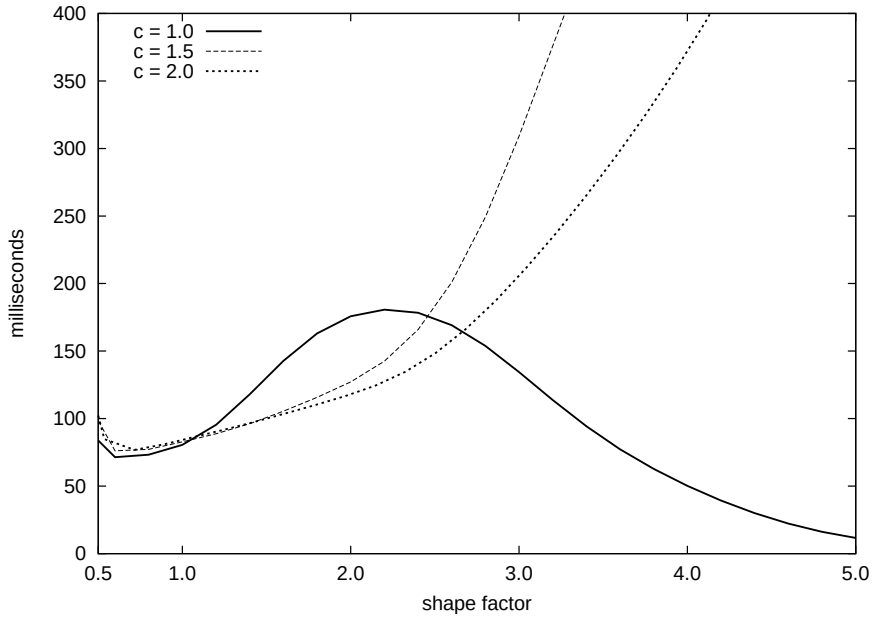
The model uses the following variables: one variable to represent the number of bins used to pack the items; one variable per item representing which bin the item is assigned to; and a variable per bin representing its load. The main constraint included in the model is the global bin packing constraint proposed by Paul Shaw [Sha04], enforcing that the packing of items into bins corresponds to the load variables. Those items whose size is greater than half of the bin capacity are directly placed into different bins. If a solution uses a number of bins smaller than the upper bound, then the load associated with unused bins is set to 0, and symmetry breaking constraints ensure that this reasoning applies to the lexicographically last variables first. Additional symmetry breaking constraints ensure that search avoids different solutions involving permutations of items with equal size.

The search strategy used is as follows. The variable representing the number of bins used in the solution is labelled first, and in increasing order, thus ensuring that the first solution found is optimal. The variables representing the item assignments to bins, and the load on each bin, are then labelled using the *complete decreasing best fit* strategy proposed by Gent and Walsh [GW97], which tries to place the items into those bins with sufficient but least free space. In our experiments a timeout of 10 seconds is used to ensure that our experiments take a reasonable amount of time. We verified that increasing this to five minutes does not significantly increase the proportion of solved instances. However, of course, for some classes a large number of time-outs were observed, so further empirical study is needed in those cases.

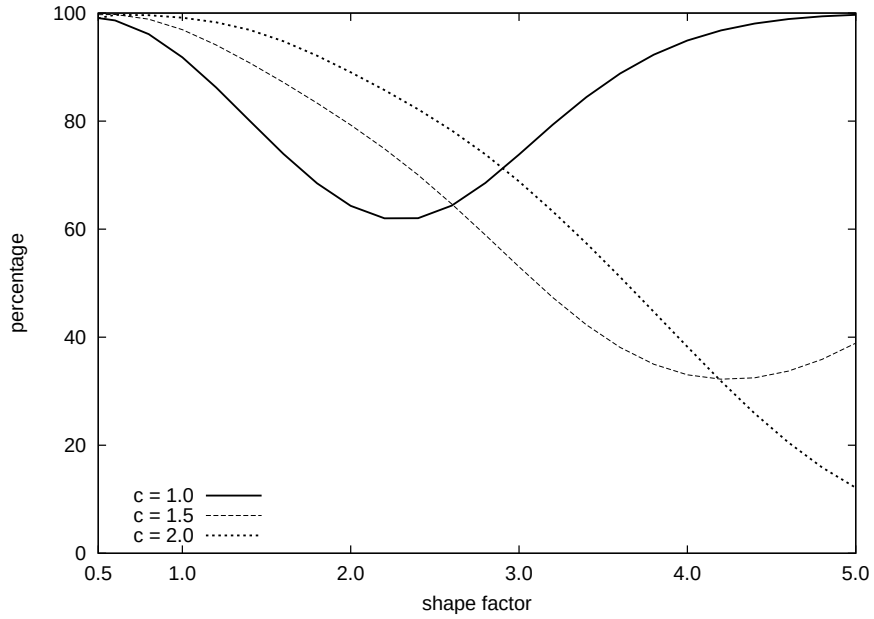
7.3.2 Small Weibull Shape Parameter Values

In this section we explore the behaviour of a systematic search on bin packing instances generated using our Weibull-based approach when considering small values of the distribution's shape parameter, specifically values ranging from 0.5 to 5.0 in steps of 0.1. Figure 7.3 presents the results – Figure 7.3a and Figure 7.3b present the average time required to those instances solved within the timeout, and the proportion of instances involved, respectively. In these plots we only consider capacity factors 1.0, 1.5, and 2.0.

It is clear that the shape factor, which defines the spread of item sizes, has a dramatic impact on the average time taken to find the optimal solution to a bin packing instance. By referring back to Figure 7.1a one can observe how the distribution of item sizes is



(a) Average running time for instances that did not timeout.



(b) Percentage of instances solved within the timeout.

Figure 7.3: Average runtime and percentage of solved instances for values of the shape parameter in the 0.5,. . . ,5.0 range.

changing. The lower values of the shape parameters correspond to distributions that have greater skew towards smaller items. As the shape parameter increases, consider value 1.5, there is a much greater range of possible item sizes. Once we get to higher shape values, consider value 5.0, the distribution of item sizes becomes more symmetric.

This shift in item size distribution impacts the difficulty of bin packing earlier when the capacity of the bin is smaller. Consider the effort required when the bin capacity is equal to the largest item, i.e. capacity factor 1.0, in Figure 7.3a. The range of shapes over which these problems are hard is quite narrow, and we shall see in the next section, that this is influenced by the bin capacity associated with the problem instance. This difficulty arises from the interaction between item size distribution and bin capacity whereby finding the best combinations of items to place in the same bin becomes challenging. As the shape parameter increases, the range of item sizes again decreases which, given the small bin capacity, makes the instance easy once more. For a bin capacity equal to the largest item size the hard region corresponds to values of Weibull shape between 1.5 and 3.0. Increasing the capacity of the bins dramatically increases the computational challenge of the problems, since again, search effort is invested in finding a good combination of items to fit into each bin. Clearly, from Figure 7.3a, we can see that problem difficulty increases as bin capacity increases.

Using our proposed Weibull-based model for generating bin packing instances we claim that not only can one model some real-world bin packing settings, as shown earlier in this chapter, but it is possible to carry out very controlled experiments on the behaviour of bin packing methods, studying the effect of the various aspects of the problem, such as bin capacity and item size distribution in isolation, or together.

7.3.3 Full Range of Shape Parameters

We have also performed a more wide-ranging study of the interaction between the shape of the Weibull distribution, bin capacity, and the hardness of bin packing for a systematic method. In this section we will briefly present a set of experiments that exhibit the various behaviours discussed above. We consider all values of the shape parameter in our data set, $0.1 \leq k \leq 19.9$. Figure 7.1b shows how the distributions with larger shape parameter values differ from those with the smaller values studied above. Essentially, these distributions have lower spread shown by successively taller density functions centering towards the value of the scale parameter.

Figure 7.4 presents both the average running time (Figure 7.4a) of the instances solved

within the timeout and the percentage of instances that this corresponds to (Figure 7.4b). The average number of bins associated with these instances is presented in Figure 7.5.

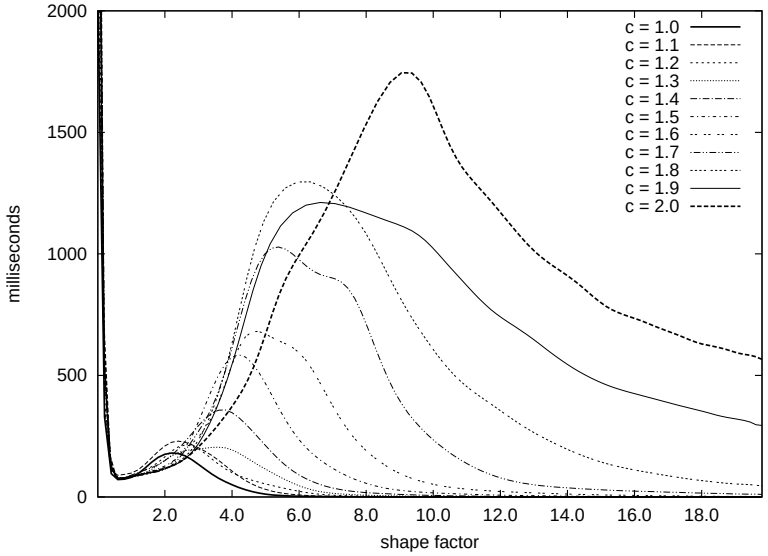
Again, in these plots we can see that, as before, problem difficulty peaks at a specific value of Weibull shape for different values of capacity (Figure 7.4). From Figure 7.5, which presents the average number of bins in an optimal solution, we can extract the average number of items per bin for each class, since all of our instances have 100 items. As before, the range of shapes over which search efforts are hard, correspond to specific ranges of numbers of bins (or average number of items per bin). Therefore, there is an obvious interrelationship between bin capacity, item size distribution, and both problem hardness and numbers of items per bin.

7.4 Bin Packing Heuristics

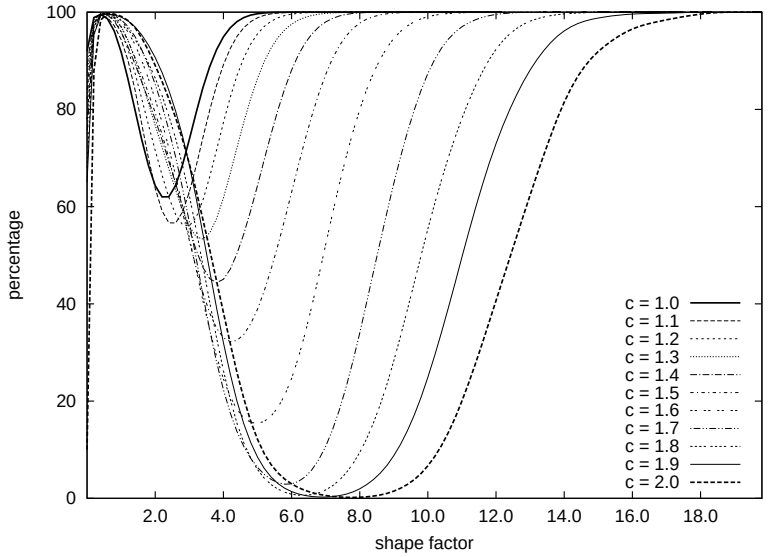
Our earlier experiments considered the performance of a systematic search method for bin packing. In this section, for completeness, we use the same set of instances to present bin packing performance when using some well-known heuristics: MAXREST, FIRSTFIT, BESTFIT and NEXTFIT. Briefly these heuristics operate as follows. MAXREST places the next item into the bin with maximum remaining space capacity; FIRSTFIT places the next item into the first bin that can accommodate it; BESTFIT places the next item into the bin that will have the least remaining capacity once the item has been accommodated by it; finally, NEXTFIT keeps the last bin open and creates a new bin if the next item cannot be accommodated in the current bin, which it will then close. For our experiments we used a publicly available implementation of these heuristics by Rieck.[5] Because our benchmark generator produces instances that have items sorted in decreasing order of size, the difference in performance between MAXREST, FIRSTFIT, and BESTFIT is very small, so we will only present results for MAXREST, while NEXTFIT will be presented separately.

Figure 7.6 presents the results, in each case showing the difference in the average number of bins as compared with the optimal value found by the systematic search method used earlier. A representative set of results for MAXREST, FIRSTFIT, and BESTFIT are presented in Figure 7.6a using MAXREST as the example, while those for NEXTFIT are presented in Figure 7.6b. Interestingly the quality of the solutions found using the MAXREST, FIRSTFIT, and BESTFIT heuristics closely follows the

[5] `http://bastian.rieck.ru/uni/bin_packing/`



(a) Average running time for instances that did not timeout.



(b) Percentage of instances solved within the timeout.

Figure 7.4: Average runtime and percentage of solved instances for values of the shape parameter for range of Weibull shapes that is sufficiently wide to exhibit the easy-hard-easy behaviour in search effort.

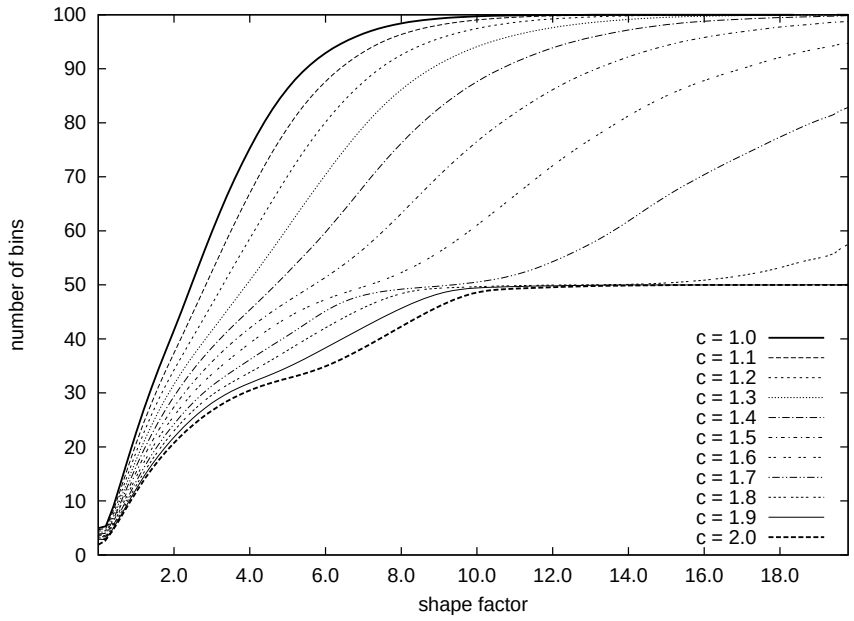


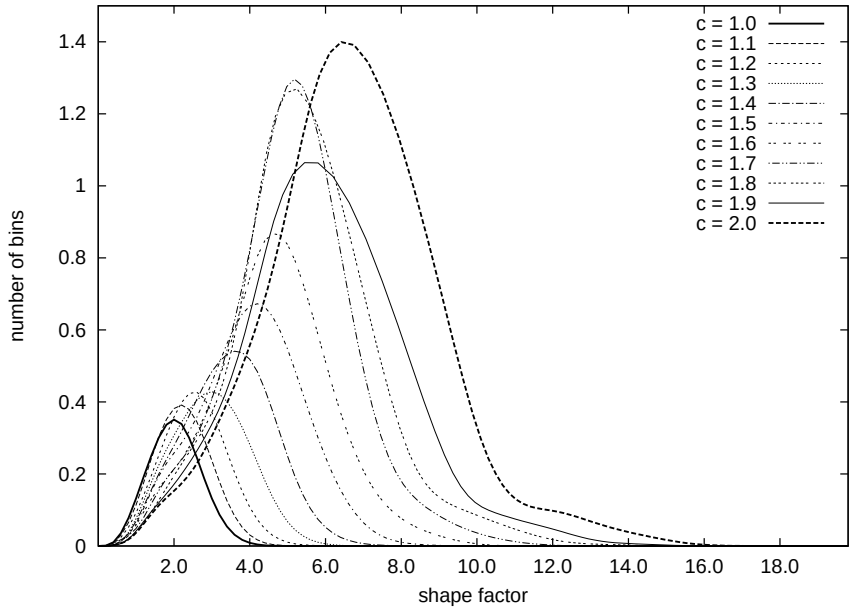
Figure 7.5: Average number of bins associated with the optimal solutions to the instances presented in Figure 7.4.

difficulty of the problem when using a systematic solver. This makes intuitive sense, since for these problems, finding a good combination of items to give a good quality solution is difficult.

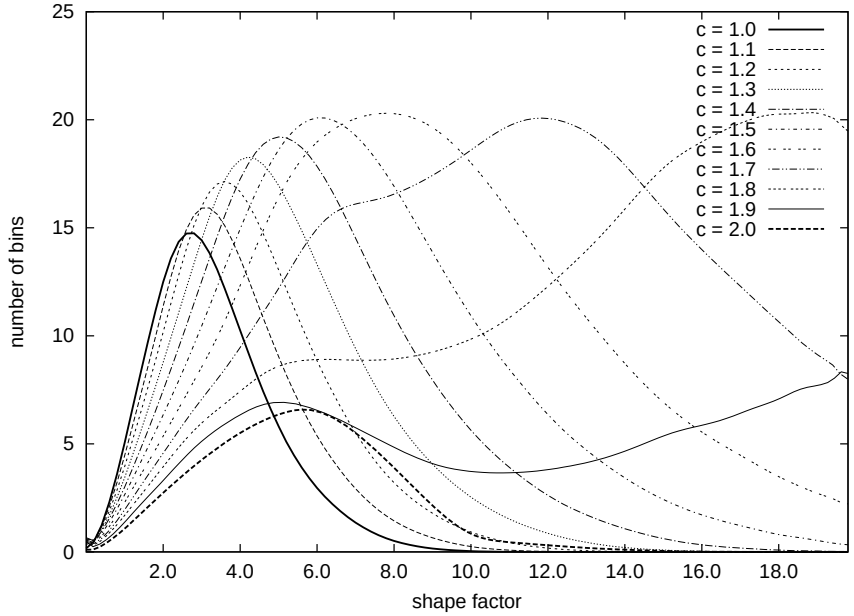
This performance contrasts starkly with that of the NEXTFIT heuristic (Figure 7.6b) which does significantly worse than optimal across almost all values of Weibull shape. While, the greediness of this heuristic does not pay off, the more considered reasoning used by the other heuristic does, except when the problem is even challenging for systematic search.

7.5 Conclusion and Limitations

In this chapter we have presented a parameterisable benchmark generator for bin packing instances based on the well-known Weibull distribution. The motivation for our work in this area comes from the domain of data centre optimisation and, in particular, workload consolidation which can be viewed as multi-capacity bin packing. We have demonstrated how our approach can very accurately model real-world bin packing problems, e.g. those from the ROADEF/EURO Challenge, and from real-world



(a) The difference in the average number of bins in solutions found using heuristics like MAXREST, FIRSTFIT, or BESTFIT. Because these perform similarly we only present results for MAXREST.



(b) The difference in the average number of bins in solutions found using NEXTFIT.

Figure 7.6: The difference in the average number of bins required by each of the heuristics and the optimal solutions - if a heuristic finds the optimal solution the difference is 0.

examination timetabling problems. We also presented an empirical analysis of both systematic search and heuristic methods for bin packing based on a large benchmark suite generated using our approach, showing a variety of interesting behaviours that are otherwise difficult to observe systematically. We observed that for all bin capacities, the number of bins required in an optimal solution increases as the Weibull shape parameter increases. However, for each bin capacity, there is a range of Weibull shape settings, corresponding to different item size distributions, for which bin packing is hard for a CP-based method.

The model we have presented here can, of course, be trivially extended to produce benchmark generators for a variety of other important problems, such as knapsacks, multi-processor scheduling, job shop scheduling, timetabling, to name but a few.

Chapter 8

Conclusions and Further Work

Summary. *This chapter concludes the discussion by first coming back to the two sub-thesis defended in this dissertation and drawing some conclusions. We then outline some possible ways the work presented in this dissertation could be extended.*

8.1 Conclusion

Throughout this dissertation we have explored various workload consolidation models and optimisation techniques to reduce the economic and ecological footprints of cloud systems. Under the assumption that more efficient consolidation translates into a more energy efficient system, we have developed a number of novel approaches to workload consolidation. In particular, we have discussed the two following claims.

Sub-thesis 1. *To date a full body of literature is focused on optimising energy consumption in cloud computing infrastructures. Many approaches rely on well-known combinatorial problems, either studied on their own or in the context of an other application. We claim that static workload consolidation can be seen as an novel variant of the bin packing problem. As such, this combinatorial problem needs to be formally described and solving approaches studied. We show that static consolidation on its own is an efficient tool for reducing resource wastage in data centres.*

Defence. Chapter 3 employed mathematical modeling to formally express a core workload consolidation problem found in many applications pertaining to cloud systems. We explored a number of techniques from the realm of operations research and constraint programming to solve this core consolidation problem in an offline context.

Chapter 4 expanded on the latter by casting it in a semi-online setting in which information is only partially known at decision time. We developed a set of solving approaches tailored to provide solutions to the consolidation problem under severe computational time limits. Although the latter aspect is often ignored in the literature, cloud providers need to implement fast decision making policies.

Sub-thesis 2. *A common issue in the daily operation of data centres is dealing with incomplete or even missing information regarding the nature of the workload that needs to be consolidated. We claim that some missing information can be at least partially retrieved with standard machine learning techniques and successfully used within optimisation processes.*

Defence. In Chapter 5, we discussed a consolidation problem for which uncertainties are coming from unknown object sizes at decision time. The specific application studied there needed to take into account object eviction to avoid over allocating machines. We explored the interactions between standard machine learning models used to retrieve missing information and the outcome of consolidation policies. In addition, Chapter 6, the nature of the uncertainty differed from previous chapters in the sense that the size of the objects is varying in time. The information available at decision time was restricted by the online nature of the problem. In this chapter we discussed a rather different consolidation problem in which workload can be migrated in order to retain a minimum energy cost while satisfying operational constraints.

8.2 Future Work

The work presented in this dissertation could be extended in a number of ways.

8.2.1 Workload Consolidation as a Component to Complex Cloud Systems

Although the focus of this dissertation was put on core workload consolidation problems, a large body of literature is focused on modeling and optimising cloud systems in a much larger scope. At the level of data centres, many challenges ranging from modeling intra- and inter-data centres networking throughput and latency [GHMP08, SLX10, PY10], to optimising non-linear and continuous models for the thermal behaviour within data centres must be tackled [CCMO15, CSG11, SC11].

We abstracted many of these aspects relevant to daily operations in cloud settings in order to focus the analyses on core workload consolidation problems. Introducing these elements within the workload consolidation models studied here would reveal that consolidation problems exhibit conflicting objectives. For instance, we provide in Chapter 4 and Chapter 5 evidence that there is a trade-off between aggressive workload consolidation and average delay in starting time of tasks. Another example of the many trade-offs that can be studied in cloud environments can be found in Chapter 6 in which we explored the balance between electricity cost and number of migrations allowed in the system.

These trade-offs should be understood and studied in terms of multi-objective problems [XF10, PB13]. Approaching the problems discussed in this dissertation from such an angle would allow decision makers to choose a given consolidation level with full knowledge of possible trade-offs.

8.2.2 Competitive Analysis

This dissertation features a variety of workload consolidation problems for which we developed online or semi-online algorithms. These approaches were empirically evaluated on various data sets extracted from real world cloud computing environments. Although these approaches have been shown to perform well, there is much scope for deriving stronger properties on their behavior through competitive analysis. Competitive analysis allows one to measure the performance gap between an online algorithm oblivious to incoming inputs and an optimal offline algorithm having access to all future inputs [Fia98]. Similar approaches could specifically be applied to the online or semi-online optimisation models discussed in Chapter 4 and 5. There is an opportunity for such analysis on algorithms discussed in these sections.

8.2.3 Understanding Prediction/Optimisation Interactions

A number of chapters in this dissertation leverage machine learning to retrieve missing or incomplete information needed to ease the decision process. This particularly the case in Chapter 4 in which statistical assumptions are made on the duration of tasks. Similarly, Chapter 5 models stochastic tasks sizes within the same context. There the use of machine learning techniques was explicitly leveraged to feed information to heuristics responsible for allocating tasks to machines.

It is in general not obvious how to characterise the link between information quality as an input to the optimisation step versus the quality of solutions returned by the optimiser [IOS12b]. As an illustration of the latter claim, we note very different behaviors in Chapter 4 and Chapter 5 respectively. In Chapter 4, the experimental section shows evidence that uncertainty on input data has a rather limited impact on the outcome of the optimisation process. In this application, there is a negative sub linear relationship between uncertainty and solution quality achieved in the optimisation step. On the other hand, in a different context, Chapter 5 shows that hard capacity constraints can be broken due to uncertain task sizes. Uncertainties in the input data is shown to have a rather large impact on the consolidation process.

Finding a systematic way to characterise the function from input data uncertainties to quality of the consolidation process could help developing business practices in such a way that the price of accessing cloud structures is a function of *a priori* information on the tasks to be scheduled.

References

[ACBO18] Vincent Armant, Milan De Cauwer, Kenneth N. Brown, and Barry O'Sullivan. Semi-online task assignment policies for workload consolidation in cloud computing systems. *Future Generation Computer Systems*, 82:89–103, 2018.

[AFG+09] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[AGH+15] Raja Wasim Ahmad, Abdullah Gani, Siti Hafizah Ab Hamid, Muhammad Shiraz, Abdullah Yousafzai, and Feng Xia. A survey on virtual machine migration and server consolidation frameworks for cloud data centers. *J. Network and Computer Applications*, 52:11–25, 2015.

[ARA14] Omar Arif Abdul-Rahman and Kento Aida. Towards understanding the usage behavior of Google cloud users: the mice and elephants phenomenon. In *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 272–277, Singapore, December 2014.

[ARGA04] Adriana C. F. Alvim, Celso C. Ribeiro, Fred Glover, and Dario J. Aloise. A hybrid improvement heuristic for the one-dimensional bin packing problem. *Journal of Heuristics*, 10(2):205–229, 2004.

[BB10a] Anton Beloglazov and Rajkumar Buyya. Energy efficient allocation of virtual machines in cloud data centers. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, CCGrid 2010, 17-20 May 2010, Melbourne, Victoria, Australia*, pages 577–578, 2010.

[BB10b] Anton Beloglazov and Rajkumar Buyya. Energy efficient resource management in virtualized cloud data centers. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 826–831, Washington, DC, USA, 2010. IEEE Computer Society.

[BB12] Anton Beloglazov and Rajkumar Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurr. Comput. : Pract. Exper.*, 24(13):1397–1420, September 2012.

[BB13] János Balogh and József Békési. Semi-on-line bin packing: a short overview and a new lower bound. *Central European Journal of Operations Research*, 21(4):685–698, Dec 2013.

[BBA10] Rajkumar Buyya, Anton Beloglazov, and Jemal Abawajy. Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges. *arXiv:1006.0308*, 2010.

[BFF+10] Peter Bodik, Armando Fox, Michael J. Franklin, Michael I. Jordan, and David A. Patterson. Characterizing, modeling, and generating workload spikes for stateful services. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 241–252, New York, NY, USA, 2010. ACM.

[BGG+10] Andreas Berl, Erol Gelenbe, Marco Di Girolamo, Giovanni Giuliani, Hermann de Meer, Dang Minh Quan, and Kostas Pentikousis. Energy-efficient cloud computing. *Comput. J.*, 53(7):1045–1051, 2010.

[BJ70] George Box and Gwilym Jenkins. *Time series analysis: Forecasting and control*. Holden-Day, 1970.

[BJK14] Sebastian Berndt, Klaus Jansen, and Kim-Manuel Klein. Fully dynamic bin packing revisited. *CoRR*, abs/1411.0960, 2014.

[BJM09] Cynthia Barnhart, Hai Jiang, and Lavanya Marla. OPTIMIZATION APPROACHES TO AIRLINE INDUSTRY CHALLENGES: airline schedule planning and recovery. In Cynthia Barnhart, Uwe Clausen, Ulrich Lauther, and Rolf H. Möhring, editors, *Models and Algorithms for Optimization in Logistics, 21.06. - 26.06.2009*, volume 09261 of *Dagstuhl*

Seminar Proceedings. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2009.

[BKA+87] Gru Brundtland, Mansour Khalid, Susanna Agnelli, Sali Al-Athel, Bernard Chidzero, Lamina Fadika, Volker Hauff, Istvan Lang, Ma Shijun, Margarita Morino de Botero, Magendra Singh, Saburo Okita, and And Others. *Our Common Future ('Brundtland report')*. Oxford Paperback Reference. Oxford University Press, USA, May 1987.

[BKB07] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. Dynamic placement of virtual machines for managing SLA violations. In *Integrated Network Management, 2007. IM'07. 10th IFIP/IEEE International Symposium on*, pages 119–128. IEEE, 2007.

[Boo] Boost: free peer-reviewed portable C++ source libraries. Version 1.47.0. `http://www.boost.org/`.

[Bre01] L. Breiman. Random forests. In *Machine Learning*, 45:1, pages 5–32. Springer, 2001.

[CC09] Jonathan D. Cryer and Kung-Sik Chan. *Time Series Analysis With Applications in R*. Springer, New York, 2009. ISBN 978-0-387-75958-6.

[CCG+13] Edward Coffman, János Csirik, Gábor Galambos, Silvano Martello, and Daniele Vigo. Bin Packing Approximation Algorithms: Survey and Classification. In Panos M. Pardalos, Ding-Zhu Du, and Ronald L. Graham, editors, *Handbook of Combinatorial Optimization*, pages 455–531. Springer New York, 2013.

[CCMO15] Danuta Sorina Chisca, Ignacio Castiñeiras, Deepak Mehta, and Barry O'Sullivan. On energy- and cooling-aware data centre workload management. In *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2015, Shenzhen, China, May 4-7, 2015*, pages 1111–1114. IEEE Computer Society, 2015.

[CCO12] Ignacio Castiñeiras, Milan De Cauwer, and Barry O'Sullivan. Weibull-based benchmarks for bin packing. In *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, pages 207–222, 2012.

[CCZ14] Michele Conforti, Gerard Cornuejols, and Giacomo Zambelli. *Integer Programming*. Springer Publishing Company, Incorporated, 2014.

[CD14] T. Chai and R. R. Draxler. Root mean square error (RMSE) or mean absolute error (MAE)?. *Geoscientific Model Development Discussions*, 7(1):1525–1534, 2014.

[CDP05] Amip J. Shah Chandrakant D. Patel. Cost model for planning, development and operation of a data center. Technical report, Internet Systems and Storage Laboratory HP Laboratories Palo Alto, june 2005.

[CENC03] Javier Contreras, Rosario Espínola, Francisco J. Nogales, and Antonio J. Conejo. Arima models to predict next-day electricity prices. *IEEE Treansaction on Power Systems*, pages 1014–1020, 2003.

[CGJ97] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for NP-hard problems. chapter Approximation Algorithms for Bin Packing: A Survey, pages 46–93. PWS Publishing Co., Boston, MA, USA, 1997.

[CJCG+13a] Edward G. Coffman Jr., János Csirik, Gábor Galambos, Silvano Martello, and Daniele Vigo. *Bin Packing Approximation Algorithms: Survey and Classification*, pages 455–531. Springer New York, New York, NY, 2013.

[CJCG+13b] EdwardG. Coffman Jr., Janos Csirik, Gabor Galambos, Silvano Martello, and Daniele Vigo. Bin packing approximation algorithms: Survey and classification. *Handbook of Combinatorial Optimization*, pages 455–531, 2013.

[CJK+99] János Csirik, David S. Johnson, Claire Kenyon, Peter W. Shor, and Richard R. Weber. A self organizing bin packing heuristic. In *Algorithm Engineering and Experimentation, International Workshop ALENEX '99, Baltimore, MD, USA, January 15-16, 1999, Selected Papers*, pages 246–265, 1999.

[CJT16] Valentina Cacchiani, Feng Jiang, and Paolo Toth. Timetable optimization for high-speed trains at Chinese railways. *Electronic Notes in Discrete Mathematics*, 55:29–32, 2016.

[CK04] Chandra Chekuri and Sanjeev Khanna. On multidimensional packing problems. *SIAM J. Comput.*, 33(4):837–851, April 2004.

[CMO+14] Milan De Cauwer, Deepak Mehta, Barry O'Sullivan, Helmut Simonis, and Hadrien Cambazard. Proactive workload consolidation for reducing energy cost over a given time horizon. In *14th IEEE/ACM Inter-*

national Symposium on Cluster, Cloud and Grid Computing, CCGrid 2014, Chicago, IL, USA, May 26-29, 2014, pages 558–561, 2014.

[CMO16] Milan De Cauwer, Deepak Mehta, and Barry O'Sullivan. The temporal bin packing problem: An application to workload management in data centres. In *28th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2016, San Jose, CA, USA, November 6-8, 2016*, pages 157–164. IEEE Computer Society, 2016.

[CMOS13a] Hadrien Cambazard, Deepak Mehta, Barry O'Sullivan, and Helmut Simonis. Bin packing with linear usage costs - an application to energy management in data centres. In Christian Schulte, editor, *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, pages 47–62. Springer, 2013.

[CMOS13b] Hadrien Cambazard, Deepak Mehta, Barry O'Sullivan, and Helmut Simonis. Constraint programming based large neighbourhood search for energy minimisation in data centres. In *GECON*, pages 44–59, 2013.

[CO10] Hadrien Cambazard and Barry O'Sullivan. Propagating the bin packing constraint using linear programming. In David Cohen, editor, *CP*, volume 6308 of *Lecture Notes in Computer Science*, pages 129–136. Springer, 2010.

[CO13] Milan De Cauwer and Barry O'Sullivan. A study of electricity price features on distributed internet data centers. In Jörn Altmann, Kurt Vanmechelen, and Omer F. Rana, editors, *Economics of Grids, Clouds, Systems, and Services - 10th International Conference, GECON 2013, Zaragoza, Spain, September 18-20, 2013. Proceedings*, volume 8193 of *Lecture Notes in Computer Science*, pages 60–73. Springer, 2013.

[CPL10] CPLEX Team. IBM ILOG CPLEX Optimizer. urlhttp://www-01.ibm.com/software/integration/optimization/cplex-optimizer/, Last 2010.

[CPW07] S. J. F. Chang, S. H. Patel, and J. M. Withers. An optimization model to determine data center locations for the army enterprise. In *MILCOM 2007 - IEEE Military Communications Conference*, pages 1–8, Oct 2007.

[CSG11] Paolo Cremonesi, Andrea Sansottera, and Stefano Gualandi. On the cooling-aware workload placement problem. In *Proceedings of the 8th AAAI Conference on AI for Data Center Management and Cloud Computing*, AAAIWS'11-08, pages 2–7. AAAI Press, 2011.

[CT12] Valentina Cacchiani and Paolo Toth. Nominal and robust train timetabling problems. *European Journal of Operational Research*, 219(3):727–737, 2012.

[dC98] J.M. Valério de Carvalho. Exact solution of cutting stock problems using column generation and branch-and-bound. *International Transactions in Operational Research*, 5(1):35–44, 1998.

[DC12] D. S. Dias and L. H. M. K. Costa. Online traffic-aware virtual machine placement in data center networks. In *2012 Global Information Infrastructure and Networking Symposium (GIIS)*, pages 1–8, Dec 2012.

[DKC12] S. Di, D. Kondo, and W. Cirne. Characterization and Comparison of Cloud versus Grid Workloads. *IEEE International Conference on Cluster Computing (CLUSTER)*, pages 230–238, 2012.

[DKC13] S. Di, D. Kondo, and F. Cappello. Characterizing Cloud Applications on a Google Data Center. *International Conference on Parallel Processing (ICPP)*, pages 468–473, 2013.

[DKF13] Sheng Di, Derrick Kondo, and Cappello Franck. Characterizing cloud applications on a Google data center. In *42nd International Conference on Parallel Processing (ICPP)*, Lyon, France, October 2013.

[DRP05] Emilie Danna, Edward Rothberg, and Claude Le Pape. Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102(1):71–90, 2005.

[DSD10] Julien Dupuis, Pierre Schaus, and Yves Deville. Consistency check for the bin packing constraint revisited. In Lodi et al. [LMT10], pages 117–122.

[DWC10] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing: Issues and challenges. In *Proceedings of the 2010 24th IEEE International Conference on Advanced Information Networking and Applications*, AINA '10, pages 27–33, Washington, DC, USA, 2010. IEEE Computer Society.

[DWF16] M. Dayarathna, Y. Wen, and R. Fan. Data center energy consumption modeling: A survey. *IEEE Communications Surveys Tutorials*, 18(1):732–794, Firstquarter 2016.

[Dyc90] Harald Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44(2):145–159, 1990.

[ES10] Nasser A. El-Sherbeny. Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods. *Journal of King Saud University - Science*, 22(3):123 – 131, 2010.

[Fal96] Emanuel Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2(1):5–30, 1996.

[Fia98] Amos Fiat. *Online Algorithms: The State of the Art (Lecture Notes in Computer Science)*. Springer, September 1998.

[FL86] D. K. Friesen and M. A. Langston. Variable sized bin packing. *SIAM Journal on Computing*, 15(1):222–230, 1986.

[FNCR11] Tiago C. Ferreto, Marco A.S. Netto, Rodrigo N. Calheiros, and Caesar A.F. De Rose. Server consolidation with migration control for virtualized data centers. *Future Generation Computer Systems*, 27(8):1027 – 1034, 2011.

[Gec06] Gecode Team. Gecode: Generic constraint development environment, 2006. Available from `http://www.gecode.org`.

[Gen98] Ian P. Gent. Heuristic solution of open bin packing problems. *Journal of Heuristics*, 3(4):299–304, 1998.

[GHMP08] 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.*, 39(1):68–73, December 2008.

[GHMP09] Albert G. Greenberg, James R. Hamilton, David A. Maltz, and Parveen Patel. The cost of a cloud: research problems in data center networks. *Computer Communication Review*, 39(1):68–73, 2009.

[GHZ13] C. Ghribi, M. Hadji, and D. Zeghlache. Energy efficient VM scheduling for cloud data centers: Exact allocation and migration algorithms. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pages 671–678, May 2013.

[GP10] Stéphane Grandcolas and Cédric Pinto. A sat encoding for multi-dimensional packing problems. In Lodi et al. [LMT10], pages 141–146.

[GW95] Gábor Galambos and Gerhard J. Woeginger. On-line bin packing — a restricted survey. *Zeitschrift für Operations Research*, 42(1):25–45, Feb 1995.

[GW97] Ian P. Gent and Toby Walsh. From approximate to optimal solutions: constructing pruning and propagation rules. In *International joint conference on Artifical intelligence*, pages 1396–1401, 1997.

[HDL11] Fabien Hermenier, Sophie Demassey, and Xavier Lorca. Bin repacking scheduling in virtualized datacenters. In Jimmy Ho-Man Lee, editor, *CP*, volume 6876 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2011.

[HI15] Ranjani C. Hemanandhini I.G. A survey on VM consolidation for energy efficient green cloud computing. *International Journal of Emerging Technology in Computer Science and Electronics.*, 19, December 2015.

[HLLz05] Zheng Hua, Xie Li, and Zhang Li-zi. Electricity price forecasting based on garch model in deregulated market. In *Power Engineering Conference, 2005. IPEC 2005. The 7th International*, pages 1–410, 2005.

[HLW11] Yufan Ho, Pangfeng Liu, and Jan-Jan Wu. Server consolidation algorithms with bounded migration cost and performance guarantees in cloud computing. In *UCC*, pages 154–161. IEEE Computer Society, 2011.

[HOO10] Emmanuel Hebrard, Eoin O'Mahony, and Barry O'Sullivan. Constraint programming and combinatorial optimisation in numberjack. In *Proceedings of the 7th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, CPAIOR'10, pages 181–185, Berlin, Heidelberg, 2010. Springer-Verlag.

[HPS01] H. S. Hippert, C. E. Pedreira, and R. C. Souza. Neural networks for short-term load forecasting: A review and evaluation. *Power Systems, IEEE Transactions on*, 16(1):44–55, 2001.

[HTF09] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Second Edition*. Springer, 2009.

[ICM+16] Jesus Omana Iglesias, Milan De Cauwer, Deepak Mehta, Barry O'Sullivan, and Liam Murphy. Increasing task consolidation efficiency by using more accurate resource estimations. *Future Generation Comp. Syst.*, 56:407–420, 2016.

[IL09] Zoran Ivkovic and Errol L. Lloyd. Fully dynamic bin packing. In S.S. Ravi and Sandeep K. Shukla, editors, *Fundamental Problems in Computing*, pages 407–434. Springer Netherlands, 2009.

[IMC+14] Jesus Omana Iglesias, Liam Murphy, Milan De Cauwer, Deepak Mehta, and Barry O'Sullivan. A methodology for online consolidation of tasks through more accurate resource estimations. In *Proceedings of the 7th IEEE/ACM International Conference on Utility and Cloud Computing, UCC 2014, London, United Kingdom, December 8-11, 2014*, pages 89–98, 2014.

[IOS12a] Georgiana Ifrim, Barry O'Sullivan, and Helmut Simonis. Properties of energy-price forecasts for scheduling. In *CP*, pages 957–972, 2012.

[IOS12b] Georgiana Ifrim, Barry O'Sullivan, and Helmut Simonis. Properties of energy-price forecasts for scheduling. In *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, pages 957–972, 2012.

[JB09] Hai Jiang and Cynthia Barnhart. Dynamic airline scheduling. *Transportation Science*, 43(3):336–354, 2009.

[JK13] Klaus Jansen and Kim-Manuel Klein. A robust afptas for online bin packing with polynomial migration,. In *Proceedings of the 40th International Conference on Automata, Languages, and Programming - Volume Part I*, ICALP'13, pages 589–600, Berlin, Heidelberg, 2013. Springer-Verlag.

[JW12] Patrick Jaillet and Michael R. Wagner. *Online Optimization*. Springer Publishing Company, Incorporated, 2012.

[JWHT13] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning with Application in R*. Springer, 2013.

[KBSW11] Jonathan G. Koomey, Stephen Berard, Marla Sanchez, and Henry Wong. Implications of historical trends in the electrical efficiency of computing. *IEEE Annals of the History of Computing*, 33(3):46–54, 2011.

[Koo08] Jonathan G Koomey. Worldwide electricity used in data centers. *Environmental Research Letters*, 3(3):034008 (8pp), 2008.

[Kor02] Richard E. Korf. A new algorithm for optimal bin packing. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada.*, pages 731–736, 2002.

[Kor03] Richard E. Korf. An improved algorithm for optimal bin packing. In *Proceedings of the 18th international joint conference on Artificial intelligence*, pages 1252–1258, 2003.

[Lar84] Harold Larnder. Or forum—the origin of operational research. *Operations Research*, 32(2):465–476, 1984.

[LC12] Zitao Liu and Sangyeun Cho. Characterizing machines and workloads on a Google cluster. In *8th International Workshop on Scheduling and Resource Management for Parallel and Distributed Systems (SRMPDS)*, Pittsburgh, PA, USA, September 2012.

[LCW11] Chinyao Low, Yahsueh Chen, and Mingchang Wu. Understanding the determinants of cloud computing adoption. *Industrial Management and Data Systems*, 111(7):1006–1023, 2011.

[LL85] C. C. Lee and D. T. Lee. A simple on-line bin-packing algorithm. *J. ACM*, 32(3):562–572, July 1985.

[LL99] Steven H. Low and David E. Lapsley. Optimization flow control, i: Basic algorithm and convergence. *IEEE/ACM Transactions on networking*, 7(6):861–874, 1999.

[LLM03] Martine Labbé, Gilbert Laporte, and Silvano Martello. Upper bounds and algorithms for the maximum cardinality bin packing problem. *European Journal of Operational Research*, 149(3):490 – 498, 2003.

[LLRL12] Jie Li, Zuyi Li, Kui Ren, and Xue Liu. Towards optimal electric demand management for internet data centers. *IEEE Trans. Smart Grid*, 3(1):183–192, 2012.

[LMT10] Andrea Lodi, Michela Milano, and Paolo Toth, editors. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 7th International Conference, CPAIOR 2010,*

Bologna, Italy, June 14-18, 2010. Proceedings, volume 6140 of *Lecture Notes in Computer Science*. Springer, 2010.

[LTC14a] Yusen Li, Xueyan Tang, and Wentong Cai. Let's depart together: Efficient play request dispatching in cloud gaming. In *13th Annual Workshop on Network and Systems Support for Games, NetGames 2014, Nagoya, Japan, December 4-5, 2014*, pages 1–6, 2014.

[LTC14b] Yusen Li, Xueyan Tang, and Wentong Cai. On dynamic bin packing for resource allocation in the cloud. In *26th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '14, Prague, Czech Republic - June 23 - 25, 2014*, pages 2–11, 2014.

[LYQ06] M. Li, B. Yu, and M.i Qi. Pgga: A predictable and grouped genetic algorithm for job scheduling. *Future Generation Computer Systems*, 22(5):588 – 599, 2006.

[Mar] Marco R. Steenberger. Maximum likelihood programming in R.

[MBS+11] Eric R. Masanet, Richard E. Brown, Arman Shehabi, Jonathan G. Koomey, and Bruce Nordman. Estimating the energy use and efficiency potential of U.S. data centers. *Proceedings of the IEEE*, 99(8):1440–1453, 2011.

[MHL+13] Aniruddha Marathe, Rachel Harris, David K. Lowenthal, Bronis R. de Supinski, Barry Rountree, Martin Schulz, and Xin Yuan. A comparative study of high-performance computing on the cloud. In *Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '13, pages 239–250, New York, NY, USA, 2013. ACM.

[MLDMD11] Jean-Baptiste Denis Marie Laure Delignette-Muller, Regis Pouillot and Christophe Dutang. *Use of the package fitdistrplus to specify a distribution from non-censored or censored data*, April 2011.

[MMS90] Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for server problems. *J. Algorithms*, 11(2):208–230, 1990.

[MT90a] S. Martello and P. Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Appl. Math.*, 28:59–70, 1990.

[MT90b] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., 1990.

[NSB+07a] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In Christian Bessiere, editor, *CP*, volume 4741 of *Lecture Notes in Computer Science*, pages 529–543. Springer, 2007.

[NSB+07b] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, CP'07, pages 529–543, Berlin, Heidelberg, 2007. Springer-Verlag.

[NW13] Alexandra M. Newman and Martin Weiss. A survey of linear and mixed-integer optimization tutorials. *INFORMS Trans. Education*, 14(1):26–38, 2013.

[Par66] D. F. Parkhill. *The challenge of the computer utility*. Addison-Wesley Professional, USA, 1966.

[PB13] Fabio López Pires and Benjamín Barán. Multi-objective virtual machine placement with service level agreement: A memetic algorithm approach. In *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, UCC '13, pages 203–210, Washington, DC, USA, 2013. IEEE Computer Society.

[PCG+10] Nicolas Poggi, David Carrera, Ricard Gavalda, Jordi Torres, and Eduard Ayguade. Characterization of workload and resource consumption for an online travel and booking site. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'10)*, IISWC '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.

[PGGM13] Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés L. Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1 – 11, 2013.

[PY10] Jing Tai Piao and Jun Yan. A network-aware virtual machine placement and migration approach in cloud computing. In *Proceedings of the 2010 Ninth International Conference on Grid and Cloud Computing*, GCC '10, pages 87–92, Washington, DC, USA, 2010. IEEE Computer Society.

[QBM+09] R. Qu, E. K. Burke, B. Mccollum, L. T. Merlot, and S. Y. Lee. A survey of search methodologies and automated system development for examination timetabling. *J. of Scheduling*, 12(1):55–89, February 2009.

[QLM12] Haiyang Qian, Fu Li, and Deep Medhi. On energy-aware aggregation of dynamic temporal demand in cloud computing. In *COMSNETS*, pages 1–6, 2012.

[QWB+09] Asfandyar Qureshi, Rick Weber, Hari Balakrishnan, John V. Guttag, and Bruce V. Maggs. Cutting the electric bill for internet-scale systems. In *SIGCOMM*, pages 123–134, 2009.

[RB] Rashid Mohammed Roken and Masood A. Badri. Time series models for forecasting monthly electricity peak-load for Dubai.

[RBW06] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Foundations of Artificial Intelligence. Elsevier, New York, NY, USA, 2006.

[Ric05] Vitto Rici. *Fitting distribution with R*, 2005.

[RKS+08] N. Roy, J. S. Kinnebrew, N. Shankaran, G. Biswas, and D. C. Schmidt. Toward effective multi-capacity resource allocation in distributed real-time and embedded systems. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 124–128, May 2008.

[RLXL10] Lei Rao, Xue Liu, Le Xie, and Wenyu Liu. Minimizing electricity cost: Optimization of distributed internet data centers in a multi-electricity-market environment. In *INFOCOM*, pages 1145–1153, 2010.

[RR11] Jean-Charles Régin and Mohamed Rezgui. Discussion about constraint programming bin packing models. In *Proceedings of the AIDC Workshop*, 2011.

[RTG+12] 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 *ACM Symposium on Cloud Computing (SoCC)*, San Jose, CA, USA, 2012.

[RWH11] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. Google cluster-usage traces: format + schema. Technical report, Google Inc., Mountain View, CA, USA, 2011.

[RWH12] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. Obfuscatory obscanturism: making workload traces of commercially-sensitive systems safe to release. In *3rd International Workshop on Cloud Management (CLOUDMAN)*, pages 1279–1286, Maui, HI, USA, April 2012. IEEE.

[Sah04] Nikolaos V. Sahinidis. Optimization under uncertainty: state-of-the-art and opportunities. *Computers And Chemical Engineering*, 28(6):971 – 983, 2004. FOCAPO 2003 Special issue.

[SBK+16] Damián Serrano, Sara Bouchenak, Yousri Kouki, Frederico Alvares de Oliveira Jr., Thomas Ledoux, Jonathan Lejeune, Julien Sopena, Luciana Arantes, and Pierre Sens. SLA guarantees for cloud services. *Future Gener. Comput. Syst.*, 54(C):233–246, January 2016.

[SC11] Andrea Sansottera and Paolo Cremonesi. Cooling-aware workload placement with performance constraints. *Perform. Eval.*, 68(11):1232–1246, 2011.

[Sch09] P. Schaus. *Solving Balancing and Bin-Packing Problems with Constraint Programming*. PhD thesis, Université Catholique de Louvain-la-Neuve, 2009.

[Sha04] Paul Shaw. A constraint for bin packing. In Wallace [Wal04], pages 648–662.

[SKJ97] Armin Scholl, Robert Klein, and Christian Jürgens. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, 24(7):627–645, 1997.

[SLX10] Yunfei Shang, Dan Li, and Mingwei Xu. Energy-aware routing in data center network. In *Green Networking*, pages 1–8, 2010.

[SMT09] C. Strobl, J. Malley, and G. Tutz. An introduction to recursive partitioning: Rational, application, and characteristics of classification and regression trees, bagging, and random forests. In *Psychological Methods*, 14:4, pages 323–348, 2009.

[SO08] Helmut Simonis and Barry O'Sullivan. Search strategies for rectangle packing. In Peter J. Stuckey, editor, *CP*, volume 5202 of *Lecture Notes in Computer Science*, pages 52–66. Springer, 2008.

[SO11] Helmut Simonis and Barry O'Sullivan. Almost square packing. In Tobias Achterberg and J. Christopher Beck, editors, *CPAIOR*, volume 6697 of *Lecture Notes in Computer Science*, pages 196–209. Springer, 2011.

[Spa03] James C. Spall. *Introduction to Stochastic Search and Optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1 edition, 2003.

[SS10] Jörg Strebel and Alexander Stage. An economic decision model for business software application deployment on hybrid cloud environments. In *Multikonferenz Wirtschaftsinformatik, MKWI 2010, Göttingen, 23.-25.2.2010, Proceedings*, pages 195–206, 2010.

[STL10] Christian Schulte, Guido Tack, and Mikael Z. Lagerkvist. Modeling and programming with gecode, 2010.

[STL12] C. Schulte, G. Tack, and M. Z. Lagerkvist. Modeling and Programming with Gecode, 2012. `http://www.gecode.org/doc-latest/MPG.pdf`.

[SW97] P. Schwerin and G. Wäscher. The bin-packing problem: A problem generator and some numerical experiments with ffd packing and mtp. *International Transactions in Operational Research*, 4(5-6):377–389, 1997.

[SW98] P. Schwerin and G. Wäscher. *A New Lower Bound for the Bin-Packing Problem and its Integration Into MTP*. Martin-Luther-Univ., 1998.

[SY08] Yongqiang Shi and Deshi Ye. Online bin packing with arbitrary release times. *Theoretical Computer Science*, 390(1):110 – 119, 2008.

[The] The R project for statistical computing. Version 1.47.0. `http://www.r-project.org/`.

[Tim02] Christian Timpe. Solving planning and scheduling problems with combined integer and constraint programming. *OR Spectrum*, 24(4):431–448, Nov 2002.

[TZWX10] Zhongfu Tan, Jinliang Zhang, Jianhui Wang, and Jun Xu. Day-ahead electricity price forecasting using wavelet transform combined with arima and garch models. *Applied Energy*, 87(11):3606–3610, 2010.

[VdBVB11] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove. An evaluation of the benefits of fine-grained value-based scheduling on general purpose clusters. *Future Generation Computer Systems*, 27(1):1 – 9, 2011.

[VGC+13] Susan V. Vrbsky, M. Galloway, R. Carr, R. Nori, and D. Grubic. Decreasing power consumption with energy efficient data aware strategies. *Future Generation Computer Systems*, 29(5):1152 – 1163, 2013. Special section: Hybrid Cloud Computing.

[Vid04] Alfio Vidotto. Online constraint solving and rectangle packing. In Wallace [Wal04], page 807.

[Wal04] Mark Wallace, editor. *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, volume 3258 of *Lecture Notes in Computer Science*. Springer, 2004.

[Wei51] Walodi Weibull. A statistical distribution function of wide applicability. *Journal of Appl. Mech.-Transactions*, 18(3):293–297, 1951.

[Wei14] Sanford Weisberg. *Applied linear regression*. John Wiley & Sons, 2014.

[Wes] Wessa, P. (2011), Free Statistics Software, Office for Research Development and Education, version 1.1.23-r7. `http://www.wessa.net/`.

[WG96] G. Wäscher and T. Gau. Heuristics for the integer one-dimensional cutting stock problem: A computational study. *OR Spectrum*, 18(3):131–144, 1996.

[WG04] Wayne L Winston and Jeffrey B Goldberg. *Operations research: applications and algorithms*, volume 3. Thomson/Brooks/Cole Belmonte Calif Calif, 2004.

[WGM+17] Mohamed Wahbi, Diarmuid Grimes, Deepak Mehta, Kenneth N. Brown, and Barry O'Sullivan. A distributed optimization method for the geographically distributed data centres problem. In Domenico Salvagnin and Michele Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming - 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings*, volume 10335 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 2017.

[WTAPB15] Andreas Wolke, Boldbaatar Tsend-Ayush, Carl Pfeiffer, and Martin Bichler. More than bin packing: Dynamic resource allocation strategies in cloud data centers. *Information Systems*, 52:83 – 95, 2015. Special Issue on Selected Papers from {SISAP} 2013.

[WTTL12] Grant Wu, Maolin Tang, Yu-Chu Tian, and Wei Li. *Energy-Efficient Virtual Machine Placement in Data Centers by Genetic Algorithm*, pages 315–32. Springer Berlin Heidelberg, Berlin, Heidel berg, 2012.

[WW11] Annegret K. Wagler and Robert Weismantel. *The Combinatorics of Modeling and Analyzing Biological Systems*, volume 10. Kluwer Academic Publishers, Hingham, MA, USA, June 2011.

[XF10] Jing Xu and Jose A. B. Fortes. Multi-objective virtual machine placement in virtualized data center environments. In *Proceedings of the 2010 IEEE/ACM Int'L Conference on Green Computing and Communications & Int'L Conference on Cyber, Physical and Social Computing*, GREENCOM-CPSCOM '10, pages 179–188, Washington, DC, USA, 2010. IEEE Computer Society.

[YSL04] H.Y. Yamin, S.M. Shahidehpour, and Z. Li. Adaptive short-term electricity price forecasting using artificial neural networks in the restructured power markets. *International Journal of Electrical Power & Energy Systems*, 26(8):571–581, 2004.

[ZCB10] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *J. Internet Services and Applications*, 1(1):7–18, 2010.